COMPUTERIZED ACQUISITION AND ANALYSES OF IMPEDANCE DATA: A SOFTWARE SYSTEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

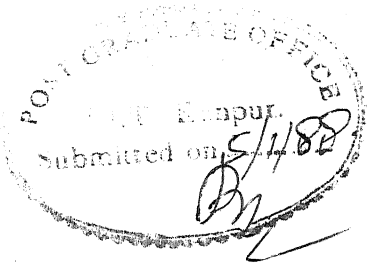
by
SANJAY BHATNAGAR

to the

MATERIALS SCIENCE PROGRAM

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JANUARY, 1988



CERTIFICATE

This is hereby certified that this work entitled 'COMPUTERIZED ACQUISITION AND ANALYSES OF IMPEDANCE DATA: A SOFTWARE SYSTEM' has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

K Shahi

[KESHAWA SHAHI]
Assistant Professor
Materials Science Program
and Department of Physics
Indian Institute of Technology
Kanpur - 208 016, India.

Date: 1.1.1988

MSP-1988-M-BHA-COM

TO PAPA AND MUMMY

ACKNOWLEDGMENTS

I wish to express my gratitude to Dr. Sanjay Gupta for his constant guidance and for being very friendly and encouraging throughout. I am also grateful to Dr. Keshaw Shahi for introducing me to the field of Lab automation and placing faith in my capabilities to be able to work in this area.

I wish to thank a friend V.C. Srivastava for his very useful tips in improving the user interface of the software.

I gratefully acknowledge the efforts taken by my colleagues Shiuli Gupta, Sujata Chaklanobis and Manoravi in using the software from very early stages of its development and constantly providing feedback which helped in its refinement.

Sanjay Bhatnagar

I wish to personally thank Dr.Sanjay Gupta, Scientific Officer B in the Advanced Cantre of Materials Science. His involvement and expertise have been vital in all stages of this project.

K Shahi

Keshawa Shahi

Thesis Supervisor

ABSTRACT

The study of A.C. conductivity is the single most common measurement in superionic conductivity research. These measurements occupy the major fraction of the experimenters time, and are subject to many inaccuracies. A software system IONICS for the automatic acquisition and analysis of A.C. Impedance data has been developed and extensively tested. IONICS consists of three modules ACQUIRE, ANALYSE and GENERATE to collect, analyse and edit data. ACQUIRE is capable of controlling both the HP4192 Impedance Analyser, and a laboratory furnace. The furnace heating and cooling can be preprogrammed with a control of better than 1.5° C using a three term control implemented through the PC. ANALYSE is used to interactively analyse the data to extract the D.C. conductivity at different temperatures and then generate the Arrhenius plots. This package has extensive graphical capabilities. This thesis discusses the IONICS system in detail. The full IONICS system requires a PC with National Instruments GPIB-PC2 and Data Translation DT28Ø5/DT7Ø7T interface hardware.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

This thesis describes the software package IONICS which has been developed for the automatic acquisition and/or analysis of conductivity data in conductivity research. The package has been optimized for the requirements of superionic conductivity research. The program has the following capabilities:

ACQUIRE module

i. Automatic control of HP4192 Impedance Analyzer
ii. Automatic control of a laboratory furnace using a chromel-alumel thermocouple.

GENERATE module

Used for creating data files identical to those produced by the ACQUIRE module manually. This module is required if data to be analyzed using the ANALYSE module, and it has not been acquired through the ACQUIRE module. It also includes a simple editor for editing data files.

ANALYSE module

i. Printing of raw data
ii. Printing of ln (σ) vs.

iii. Printing of z cos θ vs. z sin θ

iv. Arrhenius plots with fitting of straight lines

v. Complex impedance plots with fitting of circles

This package is far more versatile than the package reported from the Philips group [Dekker and coworkers, 1987]. Their system uses a non-linear least squares parameter algorithm for estimation of the parameters of an electrical equivalent circuit. The algorithm used by them does not have capabilities of online control of the furnace and the quality of fitting also appears to be inferior to that achieved in the current work.

1.1 ADVANTAGES OF AUTOMATION

A large amount of work in a laboratory is of a routine nature. If the time spent in various activities in a laboratory was to be analyzed it shall be found that quite often the bulk of it is spent in the collection and analysis of data in a fairly standardized manner. Such operations are very amenable to computer control, and offer vast returns for the initial efforts put into automating the data acquisition and analysis. Automation of these measurements and the subsequent analysis of data makes for greater efficiency in terms of effort as well as time.

In superionic conductivity research the study of A.C. impedance with frequency and temperature occupies a major fraction of the experimenters time. This time can be gainfully utilized for other purposes, if the acquisition process was to be automated.

The use of the HP4192 impedance analyzer is almost universal in this field. Thus any laboratory automation system developed is likely to find use in a large number of laboratories with minimal modifications. Furthermore, A.C. conductivity is a commonly measured quantity in many other areas of condensed matter research. With suitable changes the same software can also be used in these laboratories.

1.2 REQUIREMENTS OF A PROGRAM

A good laboratory automation program should satisfy the following criteria:

i. It should be easy to use. This requires that the program should operate in a manner akin to the thinking of the experimenter. It must have extensive and helpful menus and protect the user from system and operator errors.

ii. It should be easy to modify. In experimental research the system is constantly evolving and this makes it necessary that the programs should be written in a modular fashion making subsequent modifications easy.

In this set of programs care has been taken to design a friendly user interface, and to develop the software in a highly modular way to permit easy alterations. The program has been extensively tested with users who have a minimal training in computer usage.

1.3 CHOICE OF LANGUAGE

The commonly used languages in scientific applications are BASIC, FORTRAN and PASCAL. For control purposes FORTH is another popular language in many parts of the world. In addition there are specialized software packages like ASYST and ILS. The specialized packages were rejected since they are extremely expensive and very few laboratories having access to these.

One criterion in deciding upon the languages was the availability of handler software for the GPIB-PC2 and DT28Ø5 cards. Software handlers for GPIB-PC were available for BASIC, FORTRAN, PASCAL (both IBM and Turbo), ASYST and ASSEMBLER. DT28Ø5 handler was available for BASIC only. The documentation of GPIB-PC was insufficient to attempt to write a software handler for any other language. In the case of DT28Ø5 this was not deemed to be a major hurdle since as long the language gave access to the input/output ports of the computer it was possible to drive these cards, as the protocols were relatively simple.

BASIC was rejected due to its non structured nature which would have made it practically impossible to program in a structured fashion. BASIC is inherently unsuitable for large programs. Departure from BASIC immediately necessitated the writing of handler software for the DT28Ø5 card. FORTRAN lacks the string handling capabilities, which are so vital in GPIB interfacing. Turbo PASCAL was selected for its extensive string handling, graphics (through the GRAPHIX TOOLBOX) and port access

facilities. Turbo also provides an excellent working environment with built in editor, extensive diagnostics etc. The development of the software handler for the DT2805 cards was a worthwhile effort, which was more than compensated by the virtues of Turbo Pascal.

1.4 CHOICE OF APPROACH

Keeping in view the normal operating method the program was developed in two independent modules, to collect and analyse the data, respectively. As there was already a large amount of data which had been collected manually, it was found desirable to have a facility to enter that data into the computer for subsequent analysis by the data analysis module. For this purpose a small program for data entry was also developed.

The integrated software (named IONICS) consisting of the two modules described above, and a simple editor, has been developed with a powerful user interface. While the user gets adequate feedback on the events taking place, the exact details of the operation are hidden from him.

1.5 OBJECTIVES

This software package has been developed with the following objectives:

i. To collect impedance data automatically with minimal user interaction. Facility is provided to provide both step and scan modes of operation for both frequency and temperature.

ii. To print the data, either in full or selectively. The selective listing can be on the basis of frequency or temperature.

iii. To graphically display the data in the commonly used formats; i.e. ln (σ) vs. 1000/T or z cos(θ) vs. z sin(θ) and then fit appropriate curves to the data if required.

The various options have been implemented as hierarchial menus in the three modules. The user selects and proceeds up the desired path interactively.

The options in the ACQUIRE module are:

i. Frequency: Sweep or spot. In the sweep mode the frequencies are selected by specifying the decade(s) to be swept. There must be at least three frequencies in any decade. The frequencies are automatically spaced equally on a logarithmic scale. In the spot mode specific frequencies as specified by the user are used. Internally all operations are in the spot mode.

ii. Temperature: Temperature control can be manual or automatic. In the manual mode the DT2805 is not required and the user initiates the measurement cycle when required. In the automatic mode there are two options - spot or sweep. In the spot mode the measurement temperatures have to be specified, while in the sweep mode data is collected at intervals as specified by the user. Provision exists for controlled heating and then holding at the set point(s) for pre-specified intervals.

The options in the ANALYSE module are:

i. Plot z cos (θ) vs. z sin(θ) at specified temperatures.
ii. Plot ln(σ) vs. 1000/T for specified frequencies.
iii. Print z cos (θ) vs. z sin(θ) at specified temperatures.
iv. Print ln(σ) vs. 1000/T for specified frequencies.
v. Print raw data

The options in the GENERATE module are:

i. Edit an ASCII file
ii. Append data to a data file
iii. To enter data in a new file
iv. Rename a file
v. Delete a file
vi. To look at the directory entries or change the current directory

1.6 STRUCTURE OF THE PROGRAM

The structure of the program is illustrated in Fig. 1.1. As is evident the only link between the various modules is through the data files. The internal structure of each module is discussed in the appropriate chapters.

All the three modules, ANALYSE, ACQUIRE, and GENERATE make the IONICS system. A small shell named IONICS has been written to the run the three modules as an integrated environment.

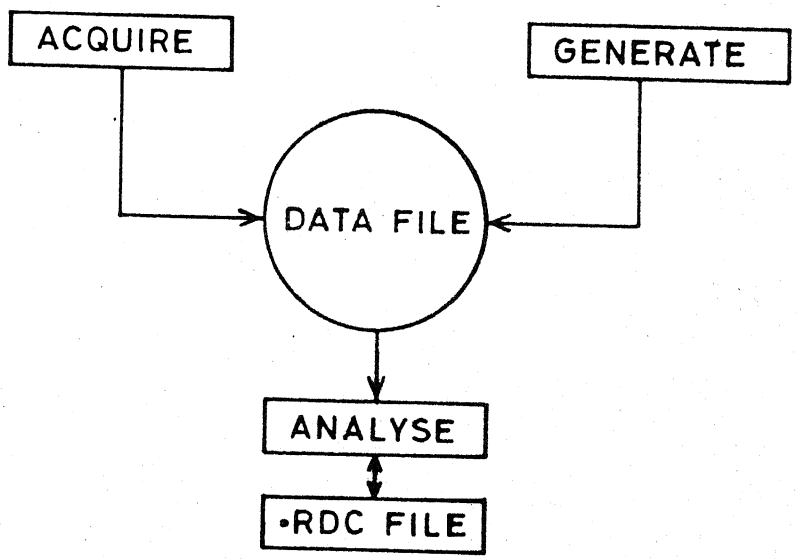


Fig.1.1 The linkage of the various modules and the data files of IONICS.

CHAPTER II

INTERFACING

An integral part of laboratory automation is connection of the measuring devices with the master controller (the computer) through an interface.

Many modern measuring devices have some intelligence of there own which does the preliminary number crunching and makes the data available for direct use by the computer. However, the computer might have to interact with the real world directly. In such a case the computer must have access to hardware which does the analog to digital conversion. The computer therefore requires two types of interfacing :

i. digital interfacing
ii. digital to analog interfacing.

While the second is done by the Digital to Analog, and Analog to digital converters (ADCs and DACs), the latter is done mostly by the GPIB interface.

2.1 GPIB Interfacing

The General Purpose Interface Bus or GPIB is a parallel digital interface bus which consist of 16 lines out of which 8 are

data/command lines, 5 are for bus management and 3 for handshaking. GPIB is also known as HPIB, IEEE488 and IEC625.

The GPIB protocol masks the local communication protocols and permits a large number of diverse devices all conforming to the same protocol to share the same bus. Thus the local formats of each device, which may or may not be different, is masked by the GPIB from the external world.

A maximum of 15 instruments can be connected to the GPIB at a time in any configuration which does not make a closed loop. The length of the connection cable between any two devices should not exceed 2 m and the total cable length should not exceed 20 meters.

GPIB communication is achieved by controlling the bus management lines and the three handshaking lines. The control for each operation is different and for that standard software drivers are available for different languages.

To start the communication, the following sequence is generally used :

i. identify the device of interest and get the GPIB address of the device. This operation is generally referred to as 'finding the device'. The GPIB address of each devices must be unique and can be set individually.

ii. initialize the device to a pre-defined state or to the state

required by the user. This is done by sending the initialization string to the device.

iii. send other command strings, if required and read the device to get the data. The data reported by each device is different and the manual of the device must be consulted for this. Generally the data comes in two formats, binary or ASCII.

Some of the GPIB commands which are implemented by controlling the bus management lines are common to all the devices. Functions are available in the device drivers for these which need to be given the device identification. All the commands which are device dependent are send as ASCII strings (some of the devices need to have ASCII strings terminated by a pre-specified terminator). These device dependent commands are listed in the device manuals.

2.2 Interfacing the HP4192 Impedance Analyzer

Interfacing with GPIB on a PC is accomplished by the DOS handler and a device driver which is specific to the language used to write the interfacing application software. The DOS handler, which comes in the file named GPIB.COM, is common to all the languages and does the disk I/O and memory management. This it does by interacting with the DOS and the ROM BIOS software. The various language interfaces provide the user with high level function which can be called by the user in the application program for I/O and GPIB communication.

For the installation of the GPIB software, an installation file by the name IBCONF.COM is provided. This gives the default implementation of the hardware, the default names for the various devices that can be connected to the bus and their corresponding addresses. Unless re-configured, the addresses of the devices should be set to one of these unused default addresses and the corresponding device name should be used in the application software. The default addresses and the device names can be changed to the requirement by running IBCONF.COM. When setting the addresses, care should be taken not to install the software for two devices with the same address and name.

Another very useful file which comes with the GPIB software is IBIC. This allows communication on the GPIB to be achieved interactively from the keyboard. This can be used to gain familiarity with the GPIB and/or a new device connected on the GPIB. Usually, before writing software to drive a new device, IBIC is used to determine the exact command sequence required by the device.

In the present work the HP4192 was interfaced to the IBM PC using the National Instruments GPIB-PC2 interface, and the Turbo Pascal GPIB handler. The handler identifies the device by a symbolic name given to the device in the installation of the handler and returns integer value which is used by the rest of the program to communicate with the device.

The to and fro communication on the bus is done by the two functions, ibwrt and ibrd. The first of these writes ASCII characters on the bus while the second accepts data from the bus. Handling of the data/commands character by character is however very difficult. Since Turbo Pascal, unlike standard Pascal, allows string handling, a small variation of ibwrt called sbwrt was written which takes entire strings.

At the beginning the presence of HP4192 is confirmed and it is then released until the furnace gets ready. Once the furnace is ready, HP4192 is set to the remote mode and the frequencies at which the data is to be acquired is fed to it one by one using the sbwrt routine. Next, the data is taken from HP4192 and if found valid, written to the disk file. Once the data for all the frequencies is gathered, it is again released till the furnace is ready for the next set of readings. This sequence is repeated until the experiment is over.

2.3 Analog and Digital Interfacing

Analog and digital interfacing involves Analog to digital converters (ADC) and the Digital to analog converters (DAC). The ADC is used to feed in the analog values to the digital devices or computers while the DAC is used to translate the digital outputs of the computers to equivalent analog values. This kind of interfacing is done to interface the computer with the analog world.

Usually, like in the case of GPIB, software interfaces are available for various languages. In case such an interface is not available writing ones own interface is not difficult, if the documentation of the card is good.

Communication with the card and its control is done by addressing the I/O ports at which the card is connected.

In the present work the DT28Ø5 ADC/DAC card was used for this interfacing. This card has an on-board 8 bit processor and is capable of various smart operations. As the Pascal interface for this card was not available, a small interface to cater to the needs of the application, was written.

The card was used with a screw termination panel (DT7Ø7T) which allows easy connections, and has a on board room temperature sensor connected to the input channel zero.

2.4 Interfacing the furnace

The furnace is interfaced to the ADC/DAC card through a solid state relay to receive the control signal from the PC and the thermo-couple to sense the temperature of the furnace is connected to channel 1 of the extension board. The solid state relay is connected to the digital I/O port Ø at bit Ø. The channel and the port numbers of the thermo-couple and the furnace are declared as global constants in ACQUIRE.PAS as t_port and f_port respectively. The bit sequence required to be outputted

to digital I/O port for the furnace depends on the bit position on the port on which the furnace is connected. The decimal value that should be outputted to put the furnace on and off is in the global constants on_ctrl_byte and off_ctrl_byte in ACQUIRE.PAS. The default is for bit Ø on port Ø. However in case the bit position is to be changed, the global constant on_ctrl_byte and off_ctrl_byte should be accordingly changed. Also if port to which the relay is connected is changed, the global constant f_port should be changed.

The furnace temperature is controlled by controlling the power delivered to the furnace. This is presently done by sending a ramp wave to put the relay on and off. The total power delivered is changed by changing the duty cycle of this ramp. Since the response of the furnace is of the order of three minutes, it acts as the integrating element. The default length of a ramp of given duty cycle is 3Ø sec. This can be changed by changing the global constant nofsteps in ACQUIRE.PAS. This is also the time interval after which the control algorithm looks at the furnace. Since the quality of control achieved is directly dependent on this, any change in this constant must be made with sufficient prior knowledge of the control algorithm used. The change in this normally will not be required. The details of the algorithms of the ACQUIRE module is given in chapter 3.

CHAPTER III

THE ACQUIRE MODULE

The Acquire module is the data generation module of the IONICS system. Apart from acquiring the data from the impedance analyser, it also permits automatic furnace control with programmed heating/cooling.

The module starts with setting up the impedance analyser. Here, there are options for automatic log sweep of the frequency range and spot frequency selection. The log sweep provided in this module is an enhancement over the log sweep mode of HP4192 impedance analyser. The number of steps per decade is user selectable here with a lower limit of 3 steps per decade. In the HP4192 sweep mode, the number of steps per decade is fixed at 20.

3.1 HARDWARE & SOFTWARE REQUIREMENTS

This module uses the GPIB interface card and the DT2805 card in case the automatic furnace control is opted for. If manual furnace control is opted for, then the DT2805 is not required. The module, cannot run without the GPIB interface card.

The GPIB-PC Turbo Pascal handler supplied by National Instruments is used as the software interface for the GPIB card. The handler for the DT2805 card has been developed and is in the

file DT28Ø5.PAS, which must be present in the same directory as the Acquire module when compiling. For details of installation, see Appendix I.

3.2 DATA ACQUISITION STRATEGY

The data acquisition can be divided into two major parts, i. control of the Impedance Analyzer, and, ii. control of the furnace.

CONTROL OF IMPEDANCE ANALYZER

The Impedance Analyzer is controlled through the IEEE488 interface. The HP4192A impedance analyzer has an averaging mode of operation which is ideally suited for ionic conductivity measurements. It should be recognized that measurements fall under two modes of operation; spot and sweep. In the spot mode data is collected at specific frequencies while in the sweep mode a number of equally spaced frequencies are used for measurements. Ideally these swept frequencies should be in a logarithmic sequence. Unfortunately, the log sweep mode in an HP4192A has only a fixed sequence of 2Ø frequencies per decade, which are too many for superionic conductivity research. Thus using the internal sweep mode is not very desirable. It is better to run the analyzer in the spot mode, and generate the measurement frequencies externally. If there are n frequencies per decade to be used then these can be calculated by:

$$f_n = f_{n-1} . 10^{1/n}$$

It is common practice to limit the starting and stopping frequencies in the sweep mode to decade values. The measuring frequencies are calculated once and then stored in an array. Having calculated the measuring frequencies it is a simple matter to take successive measurements at these frequencies. It takes the Impedance Analyzer less than 1 second to make a measurement. Usually, one makes about 20-30 measurements at a given temperature. Thus, the time taken for a set of measurements is very short. The impedance analyzer is reinitialized before each set of measurements, and the control returned to the local mode (i.e. front panel enabled) after the completion of the measurement. This makes it possible to use the analyzer for other work while the furnace is not ready. This also protects the analyzer settings from getting upset due to noise pick up etc. during the course of a long experiment.

Since, all communications through the IEEE488 interface are in the form of ASCII strings, it is very simple to get the data and write it on file. One thing which should be kept in mind is that, some languages, including Turbo Pascal do not like a + sign before positive numbers and appropriate steps should be taken to filter these out. Our system also displays the data as it is being written into the data file. During data acquisition it is necessary to check for errors in data acquisition, and take appropriate action.

TEMPERATURE CONTROL

The control of a furnace simultaneously with an impedance analyzer presents some interesting problems. A proper control algorithm becomes necessary for good stability and control. The user should have the option of both spot and sweep modes. However, unlike the case of frequencies the temperature steps can be kept equal. For good control and to protect the furnace from thermal shocks provision for programmed heating (or cooling) of the furnace is required. It is desirable to provide for two heating rates, one from start to the first measuring temperature and the second thereafter. It is also desirable to allow a holding or settling time at each measurement temperature, during which the furnace settles down at the set temperature. The minimum parameters required for efficient control of a furnace therefore are

i. Heating rate from start to first set point.
ii. Heating rate between the set points.
iii. Holding or stabilization time at each set point.

A simple on-off type of control does not work well. Furnaces under on-off control are prone to wild fluctuations specially at lower temperatures. Due to these oscillations it is quite common in manual measurements to allow many hours for the furnace to stabilize before taking measurements. A much better approach is to use Proportional-Integral-Derivative (PID) Control, also known as three term control. This technique is quite standard in

process control. The output of a PID controller, c(t), can be expressed in terms of the loop gain K_c, and the offset, c_s, as

$$c(t) = K_c e(t) + (K_c/t_i) \int e(t)dt + K_c t_d (de/dt) + c_s$$

where T_i and t_d are the integral and derivative time constants, and e_t is the error at time t. The derivative term anticipates the trend and thus permits a higher loop gain, while the integral term uses the past history to remove offsets. The incremental or recursive form of the PID equation gives the n^{th} output c_n as

$$c_n = c_{n-1} + K_c[(e_n - e_{n-1}) + (T/t_i)e_n + (t_d/T)(e_n - 2e_{n-1} + e_{n-2})]$$

The main problem in implementing the PID equation is to estimate the loop gain K_c and the time constants t_i and t_d. Following the Process Reaction Curve method of Cohen and Coon (1953), Stephanopoulos (1984) gives the following estimates for the PID control parameters.

$$K_c = (1/K)(t/T_d)\{(4/3) + (T_d/4T)\}$$

$$t_I = T_d [\{32 + (6T_d/T)\} / \{13 + (8T_d/T)\}]$$

$$t_d = T_d [4 / \{11 + (2T_d/T)\}]$$

where $K = (\text{output} / \text{input})_{\text{steady state}} = B/A$

$T = B/S$

S = slope of response at the point of inflection

T_d= time delay for the system to respond

The inherent assumption is that the actual response of the system can be approximated by a linear term and a dead time. These settings are only a good initial estimate and are supposed to be fine tuned experimentally.

The response of a typical wire wound laboratory furnace (1ØØmm dia. tube with 45Ømm winding length) operating at 1.6kW is given in Fig. 3.1. The short term response is given in the inset. The main figure is used for estimating B and the short term response for T_d and S. The response is very sluggish with a delay time of 181 seconds, i.e., the effect of a stimulus starts affecting the thermocouple after a delay of three minutes. However, using PID without any fine tuning of the Cohen-Coon parameters, using heating rates of up to 4° C per minute it was possible to achieve controlled heating with the furnace tracking the set point within ±3 degrees. The holding time required to stabilize within ±1.5° C of the set point was less than 15 minutes. For temperatures above 3ØØ° C the holding time could be reduced to 1Ø minutes. Thus it is evident that the time lost in controlled heating is amply compensated by the reduction in holding time.

For control by the PC the control signal has to be converted to a time proportioning signal. For this the fractional power output required is used to switch the furnace on and off for that fraction of time. A sampling interval of 3Ø seconds was found to work very well. The output signal was scaled by the number of

steps (3Ø in this case with the drive being every second) and the furnace kept on accordingly. A signal of ≥3Ø indicating full on and one of ≤Ø indicating full off. As the power can only go from Ø to 1ØØ% (there being no provision for auxiliary heating or forced cooling), the fractional power was also limited in the range Ø-1. This is necessary to prevent problems of latching.

3.3 PROGRAM STRUCTURE

The Acquire module constitutes one of the three modules of the IONICS system. The flow diagram for ACQUIRE is given in Fig. 3.2. The routines can be divided into three groups: set up, impedance analyzer control and temperature control. Of these the first two are always used while the third is bypassed in case temperature control is not required or done externally.

The set up section contains the main user interface and sets up the system for measurement, through an hierarchial menu structure. A value of Ø in response to the prompts of any of the menus, returns to the previous menu level. This makes it easier to correct errors in setting up. There are three parts which set up the Data File, Impedance Analyzer and, Furnace Controller respectively. They are described as under:

SET UP DATA FILE

This small routine displays the opening message and prompts the user for the name of data file to be created. If the data file

specified already exists, it gives an error message and asks for a new file name.

The user must make sure before the start of the experiment that enough space exist on the disk to store all the data. It is advisable to store the data on the floppy rather than the hard disk since in case of a power failure when data is being written to disk, the disk might get corrupted. Care should also be taken in selecting the frequency range and the number of steps per decade to avoid generation of excessive data.

SET UP IMPEDANCE ANALYZER

This module gives the user the option of using either spot frequencies or doing a logarithmic sweep measurement. In the spot mode the user is required to specify all the frequencies. In the sweep mode, the module allows the start and spot frequencies to be selected from within the seven decades of the HP4192 (1ØHz to 1ØMHz). The number of steps per decade must also be specified. The actual spot frequencies to be used are calculated internally.

SET UP TEMPERATURE CONTROLLER

Temperature control can be either manual or automatic, as mentioned earlier. In the manual mode the user keys in the temperature and then start the measurements when required. If the manual mode is selected the rest of the module is inoperative. In the automatic mode, the options of spot or sweep modes again

exist. In the spot mode the user is required to specify beforehand all the temperatures at which the measurements are to be made, along with the heating (or cooling) rates, and the holding time. In the sweep mode the user has to specify

i. start temperature
ii. stop temperature
iii. number of temperature steps
iv. initial and final heating rates
v. holding time

Internally the module always works in the spot mode. Starting from the current temperature the module heats (or cools) the furnace at the given rate and then takes measurements at the specified temperatures after allowing the furnace to stabilize for the given holding time. The measurements are taken after every temperature interval specified in the step size parameter for the furnace control.

After the various measurement parameters are set up the program uses two major modules, the Impedance Measurement Module and the Temperature Control Module.

IMPEDANCE MEASUREMENT MODULE

This module controls the HP4192 Impedance Analyzer. The operation of this module is evident from Fig. 3.3. This module is called by the Temperature Control Module. After initializing

the impedance analyzer to the spot averaging mode, it repeatedly specifies the frequencies and gets the impedance and phase angle from the analyzer, which are then displayed and written to the data file. At the end of the sequence at any given temperature it releases the impedance analyzer to the local mode.

TEMPERATURE CONTROL MODULE

As discussed earlier control is done through a PID algorithm. The setpoint is calculated every 30 seconds (this can be easily changed) and the errors used for PID are adjusted accordingly. Once the furnace goes into the holding mode the setpoint is not changed until measurements are completed. Control is implemented through a Data Translation DT2805 Low Level Analog and Digital Input-Output Card. The digital output card is connected to the furnace through a solid state relay. During the measurement phase the PID algorithm is disabled and power maintained at the level just before the start of measurement. Since measurements are over in about 20 seconds, the furnace has stabilized beforehand and the response time of the furnace is quite long, the temperature does not change appreciably during this period.

To reduce the errors due to noise in temperature measurements sixteen readings each of the thermocouple and room temperature sensor (on the DT707T screw termination panel) are taken. After sorting only the median eight are used for the actual temperature calculations. The Chromel-Alumel thermocouple is linear within the range of interest and thus temperature conversion is simple.

During the entire process the status of the system is displayed on the monitor [redacted]. The data window indicates the mode of operation of the furnace during heating (or cooling) and holding, while the data scrolls in it during measurements. Provisions exist for interrupting the program at any stage. This interruption can be to either make an orderly exit from the program or to change the parameters, i.e. set point(s), holding time, and heating rates. This was found to be more convenient and flexible than allowing for multiple heating programs to be given at the beginning.

3.4 MENU STRUCTURE

All the menus are tree structured and a return to any lower level menu is possible in a stepwise manner. This allows great flexibility and ease of error correction in setting up of the parameters.

The first menu displays the starting message indicating the name of the module which is being loaded and then setting up the a valid data file.

The menus of the impedance measurement module sets up the parameters for the measuring device and passes the control to the temperature control module.

In the temperature control module, as mentioned earlier, the options for manual or automatic temperature control exists. If

he auto mode is opted for, the above mentioned furnace arameters are set up interactively through various menus. iving a value $\leq \emptyset$ is interpreted as an indication to go to the revious menu level. This is done because a value of zero for ny of the parameters is either not a valid value or is not seful. Once the experiment starts, the program accepts no nputs from the keyboard except the ^C, ^S and the interrupts of the program, namely ^F1 for exit and ^F1Ø for edit. The edit interrupt stops the experiment and the control is transferred to the furnace control module. Any desired change in the parameters is then possible. If no change is desired hitting RETURN for every value will leave the previous values unchanged. The program will continue onwards from the point of interrupt. As far as possible, exit from the module should be from the proper routes provided (either by going down the menus till exit is provided or by the use of the ^F1 key if in the running stage). An exit through the ^C handler is to be avoided.

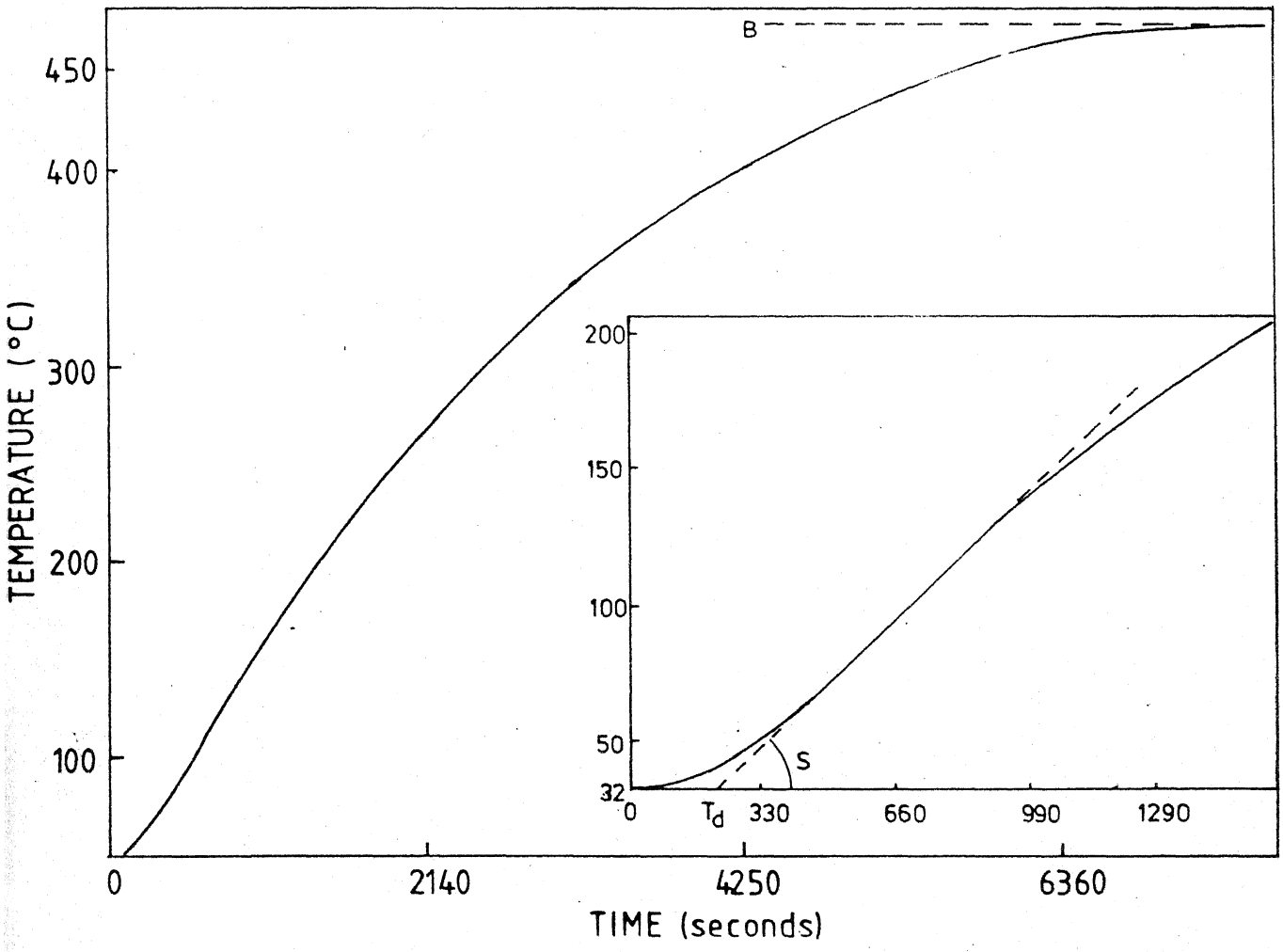


Fig.3.1. Response of a furnace to a 25% power step (inset gives the short term response). PID parameters are estimated from these curves.

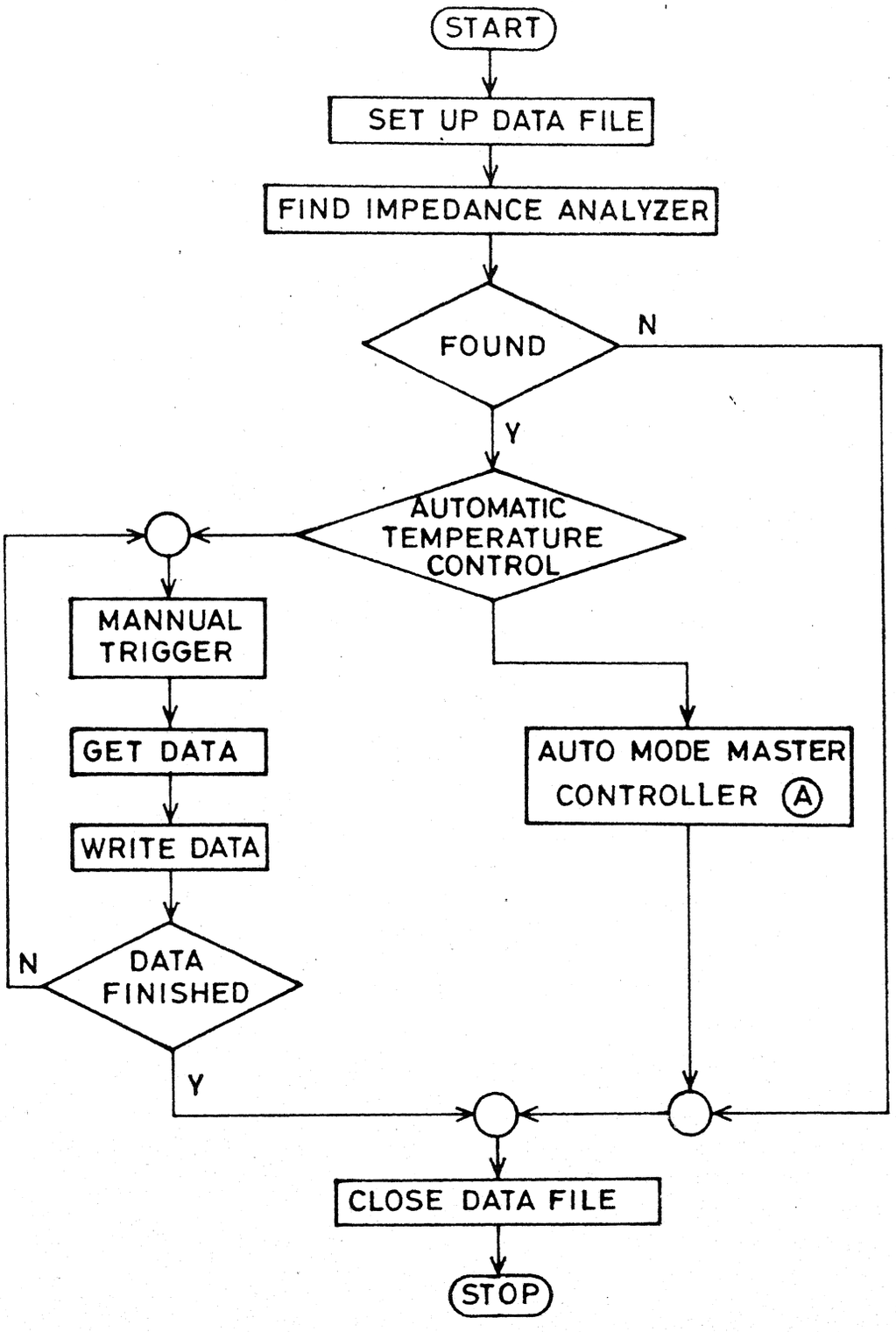


Fig.3.2 Flow-diagram of the ACQUIRE module.

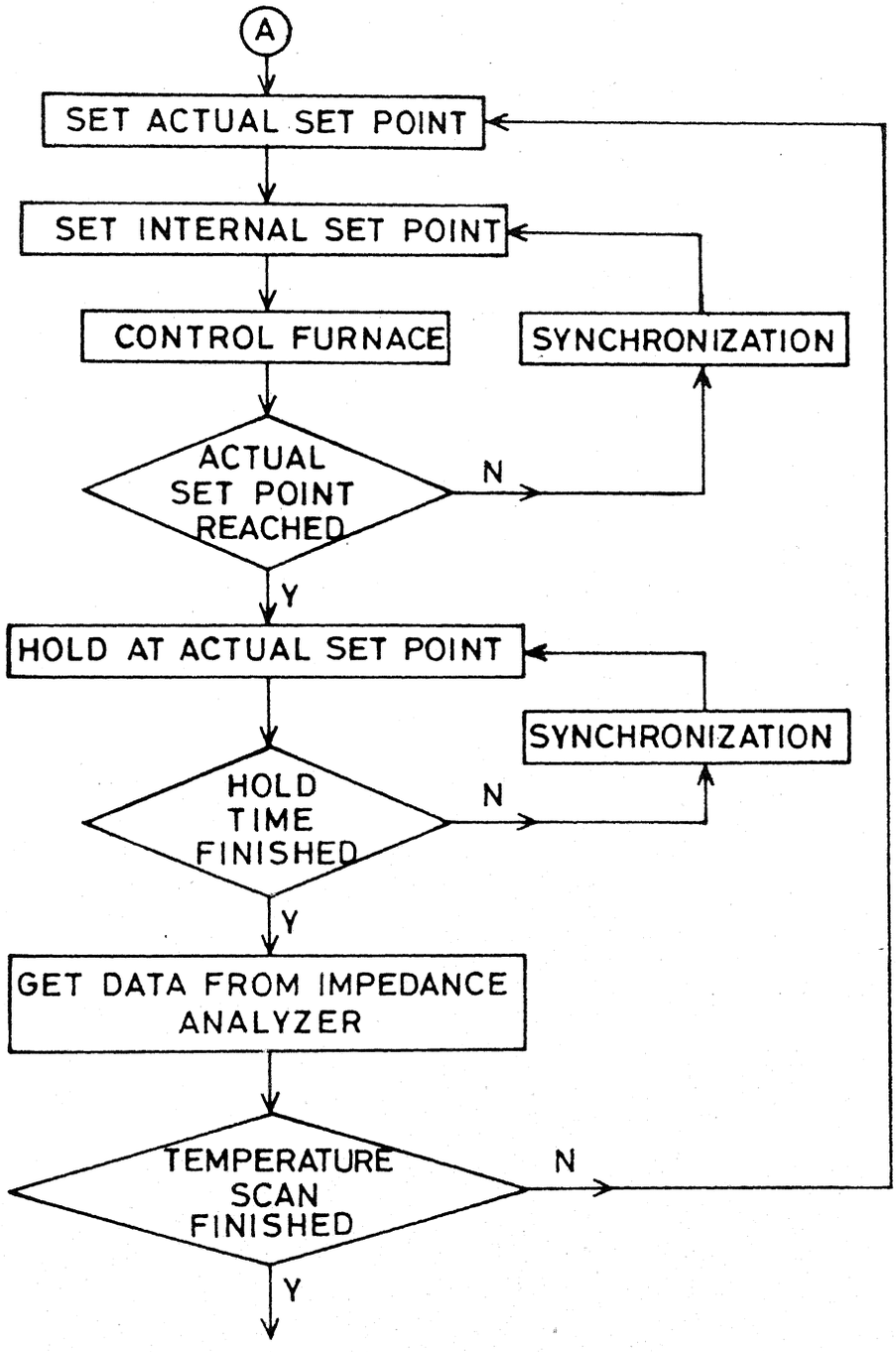


Fig.3.3 Flow diagram of the automatic mode Master Controller.

CHAPTER IV

THE GENERATE MODULE

The GENERATE module has been developed primarily to allow the creation of a data file compatible with the package. It can also be used for editing of existing data files. This module is a simple editor which is adequate for the generation and manipulation of data files for subsequent use by ANALYSE.

4.1 DESIGN

The module is driven by a tree structured menu which gives the user the control on the flow of the program. The selection is done by typing the number of the option displayed in each menu. On loading the file it presents the opening menu, which asks for the mode of file operations. The following choices are available:

1. Edit
2. Append
3. Enter
4. Rename
5. Delete
6. Directory/New directory

After selecting the mode of operation the user is prompted for the data file name, which is validated before proceeding. In case

the option chosen cannot be applied to the chosen file (like editing/appending a file which does not exist or exercising the enter option on an already existing file), then an appropriate message is displayed. In mode 1 the user is asked to define the header for the file. This text string in turn appears as the heading in all listings and plots, generated by ANALYSE. Then in mode 2&3 the type of data to be entered is requested for. The data structure of IONICS data files is as follows:

Line 1 File header or left blank for no header

Line 2- <Temperature °K> <Frequency kHz.> <Z kΩ> <Θ °>

The module has a primitive error handler, which handles any error during writing the data to the file. It signals the user by a beep and a message on the screen.

4.2 IMPLEMENTATION

The Turbo Pascal program GENERATE.PAS is accessible through the compiler or in the compiled form through GENERATE.COM. In options 1,2&3 the user is asked for the file name on which the chosen option is desired. Hitting RETURN without giving the file name exits to the previous menu. When exit is made from the main menu, the control is transferred to the parent program through which this module was loaded.

If the module is evoked from the shell (known as IONICS), it may not have enough memory to load very long files. If the memory

available is not sufficient for the entire file to be loaded a message will be displayed and editing will be allowed on the portion of file which could be loaded. In case long files are to be edited, evoke the module from DOS. For very long files a normal editor line EDLIN or WordStar (in the N mode) may have to be used.

OPTION 1:

In mode 1 the editing of the file is done by the line numbers which are displayed with every line on the screen. The menu for this mode appears on the last line of the screen and the desired option is chosen by hitting the key indicated by a letter in front of each of the options. Each of the options are described below:

Save: This is used to save the contents of the loaded file. The file is saved in the same name and no back up files are created.

View: This is the option for displaying the text of the file being edited. Upon hitting the 'V' key the prompt for the line number appears. The various operations available are as follows:

- if one number is entered then 20 lines starting form the indicated line are displayed. If 0 is entered the last line of the file is displayed. If -1 is entered 20 lines from the beginning of the file are displayed.

- if two numbers separated are entered then the second number is treated to be the number of lines to be displayed starting from the line indicated by the first number.
- if zero is entered as the second number then the file is displayed from the chosen line to end of the file.
- if no entry is made then 20 lines of the file are displayed starting from the last displayed line.
- if any of the entries is invalid then no action is taken.

Edit: This option is used to edit a line. The line number can be chosen by replying to the prompt for the same. If the chosen line number exist, the line number comes on the screen. If text is to be inserted or deleted, the cursor should be taken to the required position in the line by pressing any of the normal keys other than 'D' and 'I' and then 'D' or 'I' should be pressed for deletion or insertion respectively.

When deleting text one character is deleted on every hit of the 'D' key after the first hit and '\' appears in place of the deleted character. To come out of the delete mode the ESC key should be pressed.

When inserting text, the first hit of the 'I' key directs the editor into the insert mode. Henceforth any entry from the key board is echoed on the screen and is inserted in the line. The maximum length of any should not exceed 255 characters. If the line exceeds this limit, all characters beyond 255 are truncated and only the first 255 characters are stored. To end the insert

section press the ESC key.

If no line number is indicated when asked for, the line chosen in the previous edit section is taken as the default. Hitting the RETURN key when the line number is displayed, displays the current contents of the line.

If the indicated line does not exist no action is taken.

Insert: This mode is used to insert a new line in the file.

The new line is inserted from the line number indicated and the new line is given the number of the chosen line + 1. To end the insert line section enter 'E' or 'e' and hit a return. If no text is entered a blank line is inserted in the position indicated.

Delete: This mode is used to delete one line or a set of lines at a time. This mode takes in two arguments out of which the second one is optional. The first number indicates the first line to be deleted while the second number indicates the last line to be deleted. All lines between the first and the last line are deleted. If no second line is indicated, then only one line is deleted i.e. the line indicated by the first number.

Quit: To quit the current edit section, hit the 'Q' key. If changes are made in the loaded file and an exit is desired without saving the edited contents, the user is informed about

this fact and is asked if the edited file is to be saved or not.

OPTION 2:

This option allows the user to append additional data to an already existing data file. In case this option is exercised on a new file, an appropriate message is displayed. The data entry proceeds as in option 3 below.

OPTION 3:

This option allows data entry for use by ANALYSE. The advantage being that the data is entered in the right format and the user need to write only two numbers (the other two fields are taken care of).

To change the value of the constant parameter the first field should be entered as zero (this has been done because a value of zero for the first field is not a valid entry as far as the ANALYSE module is concerned). The user is then asked to enter a new value.

To end the data entry section and close the file, enter zero for both the fields. The module when run in this mode, does auto saving after every 10 lines.

OPTIONS 4 & 5:

Options 4 and 5 are for renaming and deleting files. They operate on one file at a time and do not take wild cards. The default directory is the current active directory.

OPTION 6:

In mode 6 there are three commands, 'D','N', and 'E'. First is for looking at the files with the mask given by the user when asked for. The second command is for changing the current directory to a new one. The new directory will be the active directory for the module till changed again. The original directory is restored as the active directory when the GENERATE module is left.

4.3 LIMITATIONS

In the current version of this module, movement of the lines of the file is not possible. Also no global replace, or search commands are implemented.

The data has to be entered in the units compatible with the ANALYSE module and in the proper order. The information of the units and the order is displayed for the users information before the data entry starts.

CHAPTER V

THE ANALYSE MODULE

5.1 Specifications and requirements

For ANALYSE an IBM PC or 100% compatible with a graphics card (Hercules or Colour Graphics) is required with 256kB or more of RAM. A numeric co-processor is desirable but not essential. If hard copies are to be taken an Epson or compatible printer is also necessary. The entire software has been developed in Turbo Pascal (Version 2.1) and uses the Turbo Pascal Graphix Toolbox (Version 3).

5.2 Hardware & software requirements

The module requires besides an IBM PC, an Epson or compatible dot matrix printer if hard copies of graphs and data are required.

The entire IONICS system is written in Turbo Pascal and requires the Turbo Pascal compiler Version 3.0 or above. Besides the compiler, it also uses the Turbo Pascal Graphix Toolbox Version 2.0 or above.

Since the software is available in the source code, it is advisable to create the .COM files as described in Appendix I, to have the integrated IONICS system working fast and efficiently.

5.3 PROGRAM STRUCTURE

The operation of the algorithm can be divided into two major branches, namely, printing and plotting of data. Printing of data is fairly straight forward and does not require any major effort. The plotting and associated curve fitting is however not straight forward. There are two kinds of plots to be catered for along with their associated curve fitting. The flow diagram of the ANALYSE module is given in Fig. 5.1.

Plotting of log(σ) vs. 1000/T requires the fitting of straight lines to the data in the above form. Once the coordinates have been calculated, a least squares fit procedure is adequate for the fitting. Provision for Arrhenius Plots must be made not only for measured frequencies, but also for DC conductivity, which is calculated from the Complex Impedance data. The d.c. resistance is proportional to the diameter of the circle fitted to the polar plot of the a.c. impedance. The most common method commonly employed is to take the span of the data as an estimate of the diameter of the circle. This is highly inaccurate, and in cases where a large part of the arc is not measured this may not even be possible.

Complex impedance plots require the plotting of z cos(θ) vs. z sin(θ) and the fitting of circles to it. However, least square fitting of a second order equation in two variables is not possible. The equation of a circle can be expressed in the form:

$$(x - a)^2 + (y - b)^2 = r^2$$

If the centre (a,b) can be estimated then a least square estimation of the radius is simple. The following approaches has been found to work extremely well in practice.

i. Estimate a guess for the centre.

ii. Establish a 5x5 grid of points around the estimated centre.

iii. Calculate the radius and then the sum of squares of the errors for each of the grid points. If j represents the j^{th} grid point and i the i^{th} data point, then

$$r_j^2 = \Sigma[(x_i - a_j)^2 + (y_i - b_j)^2] / N$$

$$e_j^2 = \Sigma(r_{i,j} - r_j)^2$$

The best fit amongst the 25 grid points is the point with the lowest sum of errors. It is by no means necessary that the range of possible centres includes the true centre. The true centre will be surrounded with points having higher sums of errors. This is achieved by the following process;

iv. If the point with minimum sum of squares of errors is in the inner 3x3 grid then it is an estimate of the centre.

v. If the point with best fit lies on the periphery then use it as the new estimate of the centre and repeat from step ii onwards.

There are two ways of estimating the starting centre:

i. For data which constitutes almost a semicircle the middle of the span of the data can be used as the first guess.

ii. For data which yields only a part of the arc, the intersection of the right bisectors of two chords, constitutes a starting guess.

In order to ensure that the initial grid includes the true centre, or is very close to it the grid points cannot be very close together. If the grid is defined to be very fine the probability of the initial guess being in the middle 3x3 grid is low. Thus, many iterations will be necessary for the requisite condition to be achieved. On the other hand a coarse grid may not yield the required accuracy.

The twin requirements of fast convergence and accuracy are met by a stepwise refinement process. Initially, a crude grid with a large spacing is defined. Once the centre is located within the accuracy of this grid the spacing is successively reduced, until the desired accuracy is obtained. It was found that a starting grid with a about 10% (of the X value) difference between each successive point of the grid seldom required more than two or three iterations to yield a good fit to most experimental data. Thereafter, the step size was successively reduced by a factor of two until the desired accuracy was obtained. This successive refinement method led to a minimum of 'hunting' in the estimation

of the centre and radius. It was possible in most cases using Turbo Pascal on a PC to achieve an accuracy of better than 1% in less than two minutes for 20 data points. With a numeric co-processor present the time was drastically reduced.

The choice between the two strategies for the centre identification is not very obvious. Experimentation with a vast amount of experimental data indicates that the first strategy is generally faster, and the second strategy may be tried only if the first failed. For ill behaved data a software switch permits changing over from one mode to the other.

5.4 MENU STRUCTURE

The nature of the module demands a very high level of user interaction. This is best achieved by a hierarchial menu structure. The first menu directs the program into the menus appropriate to the mode selected. At every stage provision exists for return to the previous level to facilitate error correction.

The opening menu (menu_0) is simply a sign on message which stays for 3 seconds. The actual working of the module starts from the second menu (menu_1), which asks for the name of the file containing the data for analysis. The file extension '.RDC' is reserved for the d.c. resistance values which are estimated by ANALYSE. Every time a file is opened for analysis a file with the same name and extension '.RDC' is automatically associated with

it, unless the file being opened has the RDC extension. To access a different data file it is essential to return to this menu.

The next menu (menu_2) displays the various options as under

i. Complex Impedance Plots (z cos θ vs. z sin θ)
ii. Plots of log (σ) vs. 1000/T
iii. Printing of z cos θ vs. z sin θ
iv. Printing of log(σ) vs. 1000/T
v. Printing of raw data

This menu directs the flow of the program to either the plotting or printing systems, beyond which control is taken by the menus of these sub-modules. An undisplayed option to plot the first two fields of the data file is also available. This is useful in general propose plotting and is used for plotting the process reaction curve required by ACQUIRE. For .RDC files options i. and iii. are disabled.

In case curve fitting is opted for, the module fits the required curves interactively. The user is required to specify the initial and the final points between which the curve is to be fitted. In case of circle fitting to the cos(θ) vs. sin (θ) plot, the accuracy can also be interactively improved up to a pre-defined limit of 0.1%. After the final fitting the fit is shown on the screen and the radius and the co-ordinates of the center of the fitted circle are displayed on the screen. In case

of the d.c. resistance (i.e., the diameter of the circle) is required for further analysis, it can be saved on the disk by replying in the affirmative to the save prompt. The resistance along with the temperature is stored on the disk in the .RDC file. The frequency and phase angles are set to zero. This file can be used for the log(σ) vs. 1000/T plot of the sample by loading this file from the file selection menu. For log(σ) vs. 1000/T plots it is possible to fit different straight lines to different regions of the plot.

It may be noted that the plots are of log(σ) vs. 1000/T instead of ln(σ) vs. 1000/T. This has been done since the Graphix Toolbox is not capable of handling log linear graphs. The scaling on the Y-axis is marked in log(σ) instead of σ. It is easier to convert log(σ) into σ than ln(σ) into σ. To convert to ln(σ) is in any case a simple matter of scaling.

Plots can be generated through screen dumps at any stage. Provision has also been made to do the same through the software for Epson printers, by answering in the affirmative to the option.

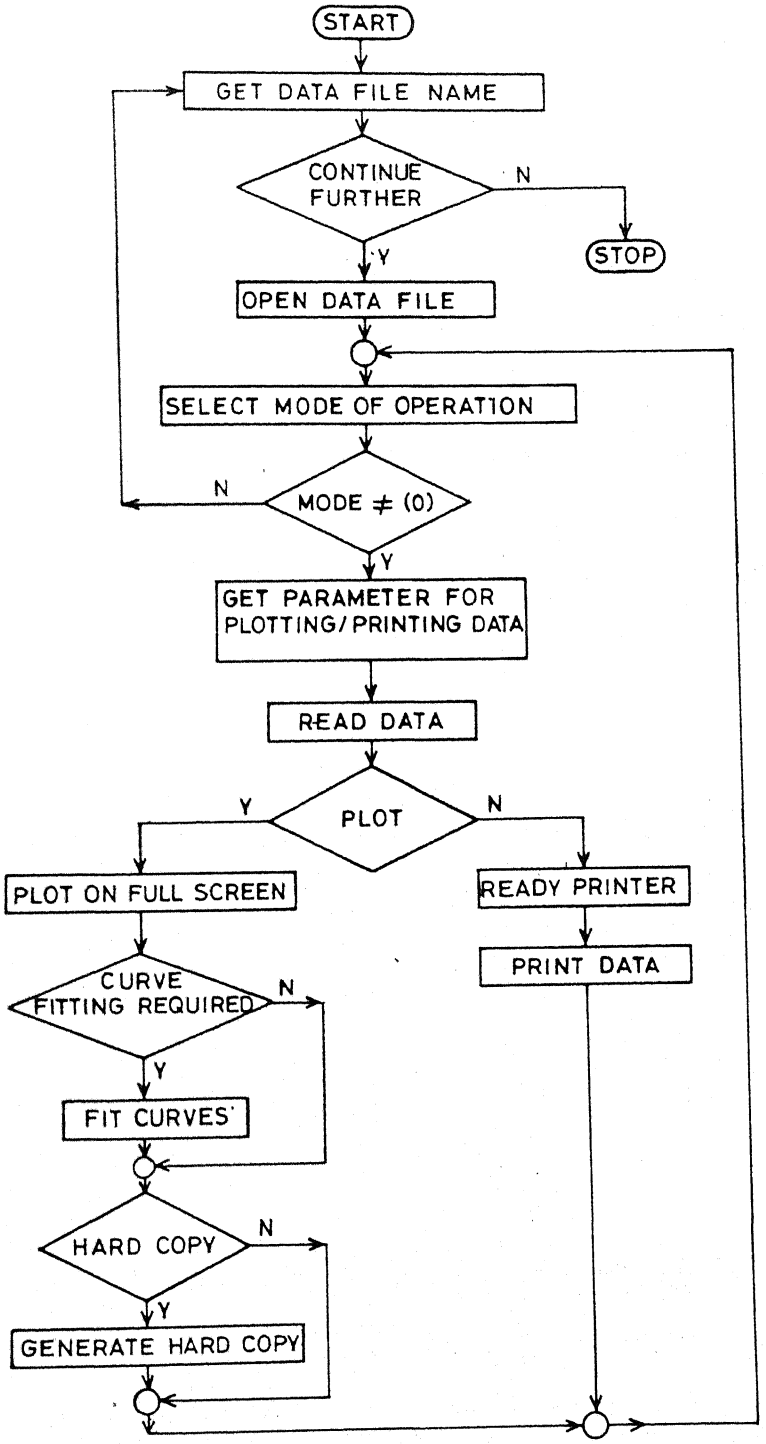


Fig.5.1 Flow diagram of the ANALYSE module.

CHAPTER VI

A SAMPLE RUN

A sample run was made to test the reliability and the practical efficiency of the IONICS system. The sample used was a composition of LiBr containing 10% Al_2O_3. The sample was first sintered at 270° C for six hours. The sample was then heated at the rate of 60 deg./hr. and readings were taken at 10°. intervals upto 437° C and subsequently at 30°. intervals upto a maximum temperature of 508° C. The holding time at each of the temperatures was 20 minutes.

Some of the complex impedance plots are given in Fig. 6.1. The Arrhenius Plot derived from the d.c. resistivity calculated from the complex impedance data is given in Fig. 6.2. It can be seen that the scatter of the points is very small. The very low scatter in the Arrhenius Plots also indicates good internal consistency and stable temperature control.

The experiment took a total time of 22 hours. Using normal on-off controllers it would have taken at least a week and the scatter of the data points would have also been much higher.

CHAPTER VI

A SAMPLE RUN

A sample run was made to test the reliability and the practical efficiency of the IONICS system. The sample used was a composition of LiBr containing 10% Al_2O_3. The sample was first sintered at 270° C for six hours. The sample was then heated at the rate of 60 deg./hr. and readings were taken at 10°. intervals upto 437° C and subsequently at 30°. intervals upto a maximum temperature of 508° C. The holding time at each of the temperatures was 20 minutes.

Some of the complex impedance plots are given in Fig. 6.1. The Arrhenius Plot derived from the d.c. resistivity calculated from the complex impedance data is given in Fig. 6.2. It can be seen that the scatter of the points is very small. The very low scatter in the Arrhenius Plots also indicates good internal consistency and stable temperature control.

The experiment took a total time of 22 hours. Using normal on-off controllers it would have taken at least a week and the scatter of the data points would have also been much higher.

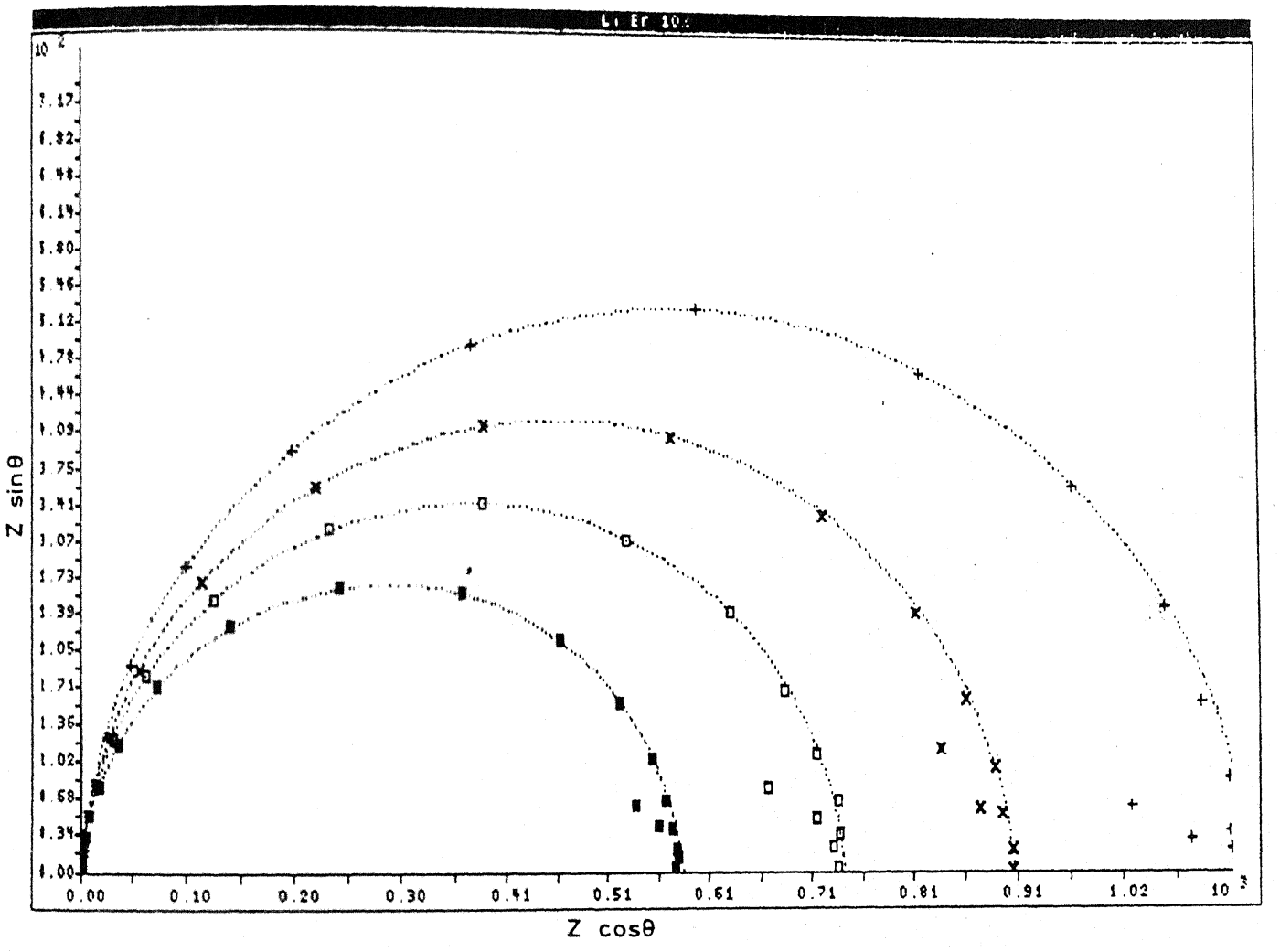


Fig.6.1 Complex impedance plots for LiBr (10% Al O) at 407, 417, 427 and 437 K.

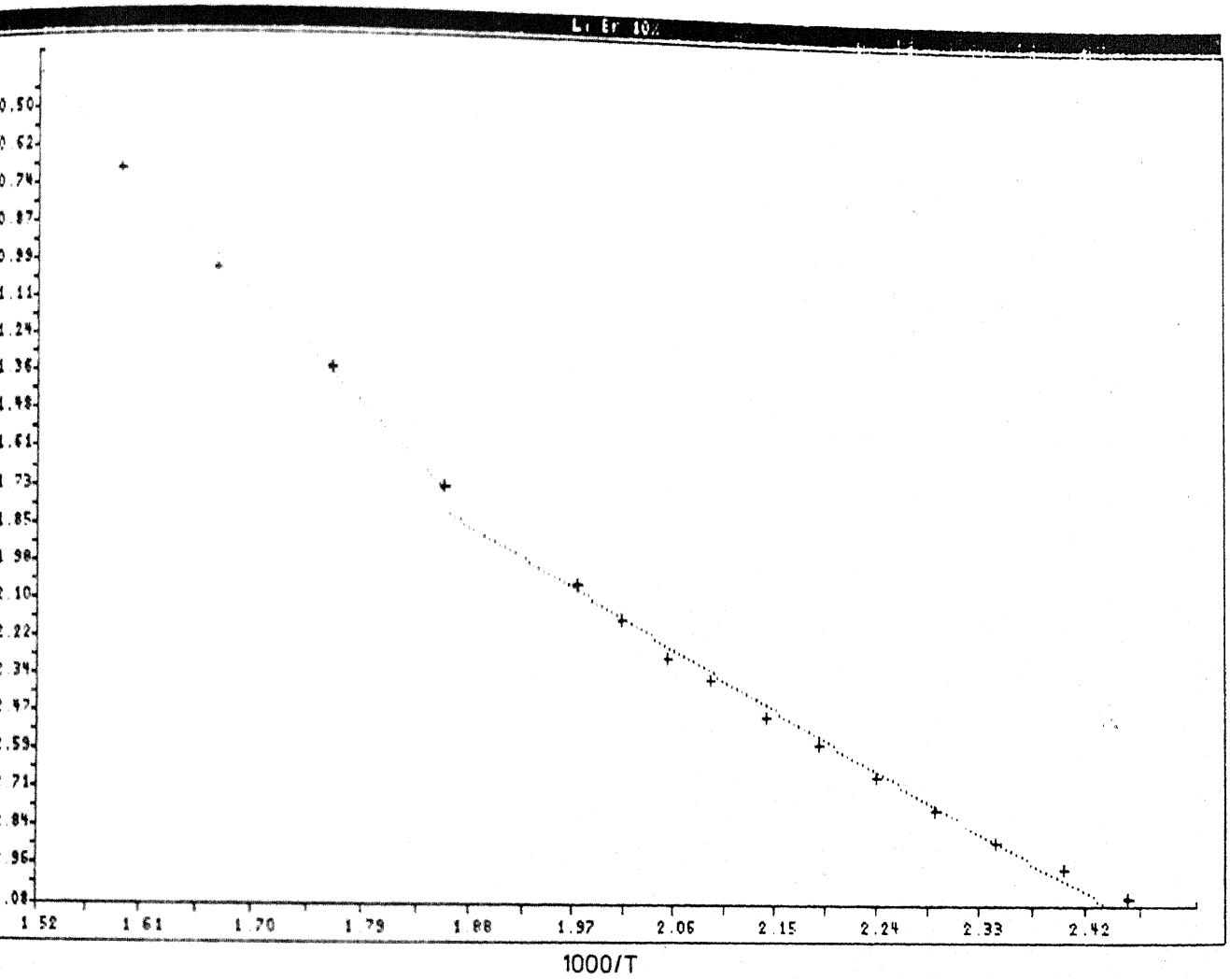


Fig.6.2 Arrehenius plot for LiBr (40% Al_2O_3) derived from the complex impadance plots.

Appendix I

Implementation of the IONICS system

The integrated IONICS system consist of three modules named ANALYSE, ACQUIRE, and GENERATE. All the three exist as files with the same name and extensions .PAS . These files contain the source code of the modules written in Turbo Pascal and need to be compiled to be run. A fourth module IONICS, which functions as a shell for these three is also available.

GENERATE module

Out of the three modules, the GENERATE module, which has a line editor and data entry function customized for the IONICS system, has no hardware dependence and can be run on any IBM PC. To run this module under Turbo Pascal, load the file GENERATE.PAS. This file may be compiled and run or better, converted into a .COM file using the 'C' compiler option.

ANALYSE module

The ANALYSE module uses the Turbo Pascal Graphix. There are two files named GRAPHIX.IBM and GRAPHIX.HGC on the Graphix Toolbox distribution diskette, and the file appropriate to the graphics card used should be renamed to GRAPHIX.SYS. The files of the toolbox used by ANALYSE are :

i. TYPEDEF.SYS

ii. GRAPHIX.SYS

iii. KERNEL.SYS

iv. WINDOWS.SYS

v. AXIS.HGH

vi. POLYGON.HGH

vii. FINDWORLD.HGH

All files listed above must be present on the same disk and directory as ANALYSE.PAS, before compilation can begin. Once the toolbox has been installed for the graphics card used, and all the above mentioned files are brought in the directory of ANALYSE.PAS, follow the procedure explained above to create the ANALYSE.COM file. At the time of running the font files of the Graphix Toolbox must be in the work directory.

ACQUIRE module

The ACQUIRE module is the most hardware dependent module of the three. The dependence is not on the basic PC itself, but in the laboratory interfaces, and the instruments used. As ACQUIRE uses the PC only in the text mode it is independent of the type of graphics hardware used. To use ACQUIRE the system must have one of the National Instruments GPIB-PC2 family of GPIB (IEEE488) interface card and software, along with an HP4192 Impedance Analyzer hooked up on it.

The GPIB card comes with a software to interface it to the DOS

and the language used in the source program (chapter II). The Turbo Pascal GPIB interface, consist of two files TPIB.COM and TPDECL.PAS. Before the module is compiled the software must be configured to address the Impedance Analyzer by the symbolic name 'HP4192'.

If automatic control of the furnace is also required a Data Translation DT28Ø5 low level interface card with DT7Ø7T screw termination panel must also be installed in the system. ACQUIRE uses the driver DT28Ø5.PAS to interface with the card. The module as set up uses the factory settings of the card. In case the IO base address is changed during installation from 2EC then the base address must be changed in DT28Ø5.PAS. A Chromel-Alumel thermocouple should be connected to input 1 and the power can be supplied to the furnace through a solid state relay connected to bit Ø of port Ø.

To compile ACQUIRE the following additional files must be present:

i. TPIB.COM
ii. TPDECL.PAS
iii. DT28Ø5.PAS

The IONICS shell

The three modules of the IONICS software can be run independently by evoking them individually from DOS (2.Ø or higher). A shell called IONICS has also been written to run the software as an

integrated system. This may be more convenient for many users. This shell is in the file called IONICS.COM and exist in the compiled form. To run the software through the shell, generate the .COM files of all the three modules and store them in the same directory as IONICS.COM. IONICS.COM is generated by compiling IONICS.PAS under the compiler option 'C' and adjusting the code and data segments to a size much larger than that generated by ANALYSE.COM. It is advised that the user should avoid recompiling IONICS unless he/she is sure of the procedure.

Note that all the .COM files of the three modules must have the names of the modules to be run from the IONICS shell. Also note that the ANALYSE module has two overlay files named ANALYSE.ØØØ and ANALYSE.ØØ1 in addition to ANALYSE.COM. These two files should always be in the same directory as ANALYSE.COM.

The IONICS shell when evoked, also makes the various programs available through a menu. Otherwise, ACQUIRE, GENERATE and ANALYSE can be run independently.

Appendix II

Estimation of PID parameters

The temperature control routines of ACQUIRE use the PID algorithm. The PID parameters used by these routines are declared as global constants. The constants are in KC_REL, KI_REL, KD_REL, NOFSTEPS and MAXT. The first three of them are the system parameters of the Cohen-Coon settings and the fourth is the one control cycle in seconds. MAXT is the estimated maximum attainable temperature. To get the PID parameters, the following procedure is used.

Cohen-Coon parameter setting

The response curve is determined by giving a step power input (25%) and monitoring the temperature as a function of time. The response curve is given in Fig. 3.1. The inset shows the short term response. The parameters as estimated are marked on the figure. The actual values calculated are as under:

MAXT = Ultimate temperature/power = 478/Ø.25 = 1912° C

Following the Process Reaction Curve method of Cohen and Coon (1953), Stephanopoulos (1984) gives the following estimates for the PID control parameters.

$$K_c = (1/K)(t/T_d)\{(4/3) + (T_d/4T)\}$$

$$t_I = T_d\ [\{32 + (6T_d/T)\} / \{13 + (8T_d/T)\}]$$

$$t_d = T_d\ [4 / \{11 + (2T_d/T)\}]$$

where K = (output / input)$_{steady\ state}$ = B/A

T = B/S

S = slope of response at the point of inflection

T_d = time delay for the system to respond

The inherent assumption is that the actual response of the system can be approximated by a linear term and a dead time. These settings are only a good initial estimate and are supposed to be fine tuned experimentally.

K = 1912° C T = 10371.53

T_d = 181 sec. and hence

t_I = 437.66 sec. and t_d = 65.61 sec.

Appendix III

Listings of programs

The listings of all the programs written for IONICS are listed here. Besides the three modules, listings of the shell IONICS and the DT28Ø5 driver are also included. For compactness, the page has been horizontally divided into two and the listings are in the compressed print.

```
PROGRAM IONICS;
  label 10;
  type name = string[15];
  var ch:integer;
      exefile:file;
      c:char;
      i:real;
      chk:byte absolute CSeg:$80;

procedure sign_on;
begin
  clrscr;
  writeln;writeln;
  writeln('
  writeln('
  writeln('
  writeln('
  writeln('

  gotoxy(1,12);
  writeln('                                  Version 1.1  October 1997');

  gotoxy(1,13);
  writeln('         Written by:                                  Supervisors:');
  writeln;
  writeln('           Sanjay Bhatnagar                            Dr.Sanjay  Gupta');
  writeln('                                                       Dr.Keshava Shahi');
  gotoxy(1,21);
  writeln('                    Advanced Centre for Materials Science');
  gotoxy(1,22);
  writeln('                    Indian Institute of Technology Kanpur');
  gotoxy(80,1);
  delay(5000);
end;

procedure drawDbox(x1,y1,x2,y2:integer);
var index:integer;
  begin
    gotoxy(x1,y1);write('r');
    for index:= 1 to (x2-x1-1) do write('-');
    write('1');
    for index:= (y1+1) to (y2-1) do
       begin
       gotoxy(x1,index);write('|');
       gotoxy(x2,index);write('|');
       end;
    gotoxy(x1,y2);write('L');
    for index:= 1 to (x2-x1-1) do write('-');
    write('J');
    gotoxy(x1,y1);
end;
```



```
procedure MASTERMENU(var choice:integer);
  var ok:boolean;io:integer;
  begin
    clrscr;
    writeln;
    writeln('
    writeln('
    writeln('
    drawDbox(1,1,45,5);
    gotoxy(6,9);writeln('OPTIONS');writeln;
    writeln('           0. Exit');
    writeln('           1. Acquire data');
    writeln('           2. Analyse data');
    writeln('           3. Generate data');
    gotoxy(42,22);
    write('Your choice [0..3] : ');
    repeat
      gotoxy(21,22);val(readkey,choice,io);gotoxy(21,22);write(choice);
      ok := (choice in [0..3]);
      if not ok then begin gotoxy(1,24);write('Invalid choice !!!');end;
    until ok;
  end;

procedure msgload(name:name);
  begin
    gotoxy(3,23);write('Loading ',name,'....');
  end;

procedure msg(name:name);
  begin
    gotoxy(3,23);write(name,' not found.');delay(1500);
  end;

  begin
    if (chk in [0..127]) then sign_on;
    10:mastermenu(ch);chk:=150;
    case ch of
      0:begin clrscr;exit; end;
      2:begin assign(exefile,'analyse.com'); {$I-} reset(exefile); {$I+}
              if ioresult <> 0 then begin msg('ANALYSE.COM');goto 10;end;
              msgload('ANALYSE.COM');
              execute(exefile);end;
      1:begin assign(exefile,'acquire.com'); {$I-} reset(exefile); {$I+}
              if ioresult <> 0 then begin msg('ACQUIRE.COM');goto 10;end;
              msgload('ACQUIRE.COM');
              execute(exefile);end;
      3:begin assign(exefile,'generate.com');{$I-} reset(exefile); {$I+}
              if ioresult <> 0 then begin msg('GENERATE.COM');goto 10;end;
              msgload('GENERATE.COM');
              execute(exefile);end
    end;
  end.
```

```
{*****************************************************************************}
{*                                                                           *}
{*                  SUPERIONIC CONDUCTIVITY DATA ACQUISITION                 *}
{*                                                                           *}
{*                        Data Acquisition Module                            *}
{*                           Sanjay Bhatnagar                                *}
{*                             August 1987                                   *}
{*                                                                           *}
{*****************************************************************************}

program ACQUIRE;

{*  Include the GPIB and DT2805 interfaces  *}

{$I TPDECL.PAS}
{$I DT2805.PAS}

label 100,200,210,300,350,400;
type
     datastr=string[10];
     tempstr=string[255];
     name=string[20];
     device=string[3];
     title=string[50];
const
     F_PORT = 0;       { port no. at which the furnace is connected }
     T_PORT = 1;       { port no. at which thermo-couple is connected }

     ON_CTRL_BYTE = 1; { byte to be written to F_PORT to put on the furnace  }
     OFF_CTRL_BYTE = 0;{ byte to be written to F_PORT to put off the furnace }

                       { The value of these bytes will depend on the bit of  }
                       { F_PORT to which the furnace is connected.  The      }
                       { values should be such that the binary bit pattern   }
                       { sets the furnace bit of the F_PORT to 1 or 0. The   }
                       { default bit is BIT0.                                }

     NOFSTEPS = 30;    { length of the control loop in sec. for furnace }
     KC_rel = 0.0137;  { kc of the Cohen & Coon settings for PID }
     TI_rel = 437.66;  { Integration time for PID }
     TD_rel = 65.61;   { Differentiation time for PID }
     MAXT = 1912.0;    { estimated max. temperature of the furnace }

     MAXFREQ = 13000.0; { Max. frequency accepted by the measuring device }
     MINFREQ = 0.005;   { Min. frequency accepted by the measuring device }

     {* Constants for GPIB Turbo Pascal interface *}

     ERR=$8000;    { IBERR value if GPIB error detected }
     TIMO=$4000;
     SRQI=$1000;
     RSC=$800;
     DATALIST='  T(°K)      F(kHz)        Z(kΩ)          θ(°)';
     DEVGLB='HP4192';  { Name of the Impedance Analyser for GPIB interface }
     CMDGLB='AIB1F1V1T3'; { Command string for the measuring device }
```



```
     CMDORG='T1';  { Command string to release the device from remote status }
var
     chk:byte absolute CSeg:$80;
     freq:array [1..200] of real;
     count,brd0:integer;
     nofdata:integer;
     fn:name;
     ch:integer;
     datfile:text;
     glberr,first,dt2805:boolean;
     tempr:real;
     cha:char;
     hd:title;
     E:array [1..3] of real;              (to hold the 3 latest mod.errors)
     initrate,frate,ME,kc,tint,tdiff:real;
     ain,edit,back:boolean;cch:char;

  function min(a,b:integer):integer;
    begin
     if a >= b then min:=b else min:=a
    end;

  function ERROR:boolean;
    begin
     ERROR:=(ierr and ibsta)<>0 ;
    end;

  procedure beep;
    begin
     sound(2000);delay(500);nosound;
    end;

  procedure drawbox(x1,y1,x2,y2:integer);

  {*  Function   :  To draw box with upper left corner at (X1,Y1)
                    and lower right corner at (X2,Y2)
      Globals used : None
      Globals mod. : None
      Routines used : None                                                  *}

   var index:integer;
     begin
      gotoxy(x1,y1);write('r');
      for index:= 1 to (x2-x1-1) do write('-');
      write('1');
      for index:= (y1+1) to (y2-1) do
        begin
          gotoxy(x1,index);write('|');
          gotoxy(x2,index);write('|');
        end;
       gotoxy(x1,y2);write('L');
      for index:= 1 to (x2-x1-1) do write('-');
      write('J');
      gotoxy(x1,y1);
     end;
```

```
procedure drawDbox(x1,y1,x2,y2:integer);

{   Function   :  To draw double box with upper left corner at
                  (X1,Y1) and lower right corner at (X2,Y2)
    Globals used : None
    Globals mod. : None
    Routines used : None                                        }

  var index:integer;
  begin
    gotoxy(x1,y1);write('r');
    for index:= 1 to (x2-x1-1) do write('-');
    write('');
    for index:= (y1+1) to (y2-1) do
      begin
        gotoxy(x1,index);write('|');
        gotoxy(x2,index);write('|');
      end;
    gotoxy(x1,y2);write('L');
    for index:= 1 to (x2-x1-1) do write('-');
    write('J');
    gotoxy(x1,y1);
  end;

procedure LOADSTR(var outst:iobuf;instr:tempstr;len:integer);

 {$ Function    :    Takes in a string and loads it in an array  $}

  var i:integer;
  begin
    for i:=1 to len do
      outst[i]:=instr[i];
  end;

procedure FINDERROR;

 {$  Function    :   Translates the error report of IBERR        $}

  begin
    glberr:=true;clrscr;
    case IBERR of
       0:begin writeln;writeln('DOS error');writeln;end;
       1:begin writeln;writeln('Function requires GPIB-PC to be CIC');
               writeln;end;
       2:begin writeln;write('The measuring instrument is either');
                       writeln(' not ON or is malfunctioning.');writeln;end;
       3:begin writeln;writeln('GPIB-PC not addressed correctly');
               writeln;end;
       4:begin writeln;writeln('Invalid argument to function call');
               writeln;end;
       5:begin writeln;writeln('GPIB-PC not System Controller as required');
               writeln;end;
       6:begin writeln;writeln('I/O operation aborted');writeln;
               glberr:=false;exit;end;
       7:begin writeln;writeln('Non-existent GPIB-PC board');writeln;end;
```



```
      10:begin writeln;writeln('I/O started before previous completed');
              writeln;end;
      11:begin writeln;writeln('No capability for operation');writeln;end;
      12:begin writeln;writeln('File system error');writeln;end;
      14:begin writeln;writeln('Command error during device call');
          writeln;end;
      15:begin writeln;writeln('Serial Poll status byte lost');writeln;end;
      16:begin writeln;writeln('SRQ stuck in on position');end
    end;
    writeln('Press any key.');
    repeat until keypressed;
    writeln;writeln('Aborting program...');
  end;

procedure ERRORSAN(code:integer);

 {$ Function  :  Translates the error reported by sources other
                 than IBERR                                     $}

  begin
    case code of
      1:writeln('Display A overflow');
      2:writeln('Display A uncalibrated');
      3:writeln('Display B overflow');
      4:writeln('Display B uncalibrated');
      5:writeln('Error in display C')
    end
  end;

function R_ASCII(value:real):datastr;

 {$  Function   :   Takes in a real variable and converts it into
                    a character string                          $}

 var temp:datastr;
  begin
      if (value)=0.005) and (value(10.0) then          {$ Formating the   $}
        begin                                          {$ char. string to $}
          str(value:9:6,temp);R_ASCII:=temp;exit;      {$ get maximum     $}
        end;                                           {$ resolution      $}
      if (value)=10.0) and (value(100.0) then
        begin str(value:9:5,temp);
          R_ASCII:=temp;exit;
        end;
      if (value)=100.0) and (value(1000.0) then
        begin
          str(value:9:4,temp);R_ASCII:=temp;exit;
        end;
      if (value)=1000.0) and (value(10000.0) then
        begin
          str(value:9:3,temp);R_ASCII:=temp;exit;
        end;
      if (value)=10000.0) and (value(=13000.0) then
        begin
          str(value:9:2,temp);R_ASCII:=temp;exit;
```

```
      end
  end;

procedure SBWRT(dev:integer;message:tempstr);

 {* Function   :  Takes in the command as a string argument and
                  sends it to the device through IBWRT function.
                  Unlike IBWRT, this routine does not need the
                  length of the command/data string          *}

  var wrt:iobuf;
      len:integer;
  begin
    len:=length(message);
    loadstr(wrt,message,len);
    ibwrt(dev,wrt,len);
  end;

procedure  WRITEHP4192DATA(tempr:real;dev:integer);

 {* Function   :  Writes the data send by HP4192, programmed for
                  format F1.  It reports impedance in kohms      *}

  label 10;
  var rd:iobuf;
      j:integer;
  begin

   window(17,15,66,22);
   ibrd(dev,rd,45);
   if error then finderror;
   case rd[1] of                                    { Check for error }
       'O':begin writeln;errorsan(1);goto 10;end;   { in the HP4192   }
       'U':begin writeln;errorsan(2);goto 10;end    { string          }
     end;
   case rd[17] of
       '0':begin writeln;errorsan(3);goto 10;end;
       'U':begin writeln;errorsan(4);goto 10;end
     end;
   if rd[33]<>'K' then
     begin
       writeln;
       errorsan(5);
       goto 10
     end;
   count:=count+1;
   writeln;                          { If no error in the HP string    }
   write(datfile,tempr:7:2,' ');     { then write the data on the file }
   write(tempr:7:2,'    ');          { and in the data window of the   }
   if rd[34]='-' then                { display                         }
     for j:=34 to 43 do
       begin
         write(datfile,rd[j]);
         write(rd[j]);
       end
```



```
   else
     for j:=35 to 43 do
       begin
       write(datfile,rd[j]);
       write(rd[j])
     end;
   write(datfile,' ');
   write('    ');
  case rd[15] of
    '2':begin rd[14]:='1';rd[15]:='5';end;   {* Conversion of the *}
    '9':begin rd[14]:='1';rd[15]:='2';end;   {* 3 value to kohms *}
    '6':if rd[13]='-' then
          rd[15]:='9'
        else rd[15]:='3';
    '3':if rd[13]='-' then
          rd[15]:='6'
        else rd[15]:='0';
    '0':begin rd[15]:='3';rd[13]:='-';end
  end;
  if rd[5]='-' then
    for j:=5 to 15 do
      begin
        write(datfile,rd[j]);
        write(rd[j]);
      end
  else
    for j:= 6 to 15 do
      begin
        write(datfile,rd[j]);
        write(rd[j]);
      end;
  write(datfile,' ');
  write('    ');
  if rd[21]='-' then                    { remove any leading '+' sign }
    for j:=21 to 31 do                  { as that is unacceptable in  }
      begin                             { Turbo Pascal                }
        write(datfile,rd[j]);
        write(rd[j]);
      end
  else
    for j:=22 to 31 do
      begin
        write(datfile,rd[j]);
        write(rd[j]);
      end;
  writeln(datfile,' ');
  10:window(1,1,80,25);
 end;

procedure clrtoxy(x,y:integer);
  var i:integer;
      xp,yp:integer;
  begin
    xp:=wherex;
    yp:=wherey;
```

```
    for i:= xp to x do
      write(' ');
    gotoxy(xp,yp);
  end;

function exist(filname:name):boolean;

{*  Function  :  To check if the file exist on disk
                 if exist then true else false        *}

    begin
      assign(datfile,filname);{$i-}
      reset(datfile);{$I+}
      exist:=(ioresult=0)
    end;

procedure writeopen(f:name;var success:boolean);

{*  Function  :  To open a file to write    *}

  begin
    assign(datfile,f);
    {$I-}rewrite(datfile);{$I+}
    success:=not(ioresult in [$51,$F0]);
  end;

procedure menu_0;

  {*  Function:     Give the opening display          *}

begin
  clrscr;drawDbox(19,6,59,15);
  gotoxy(34,7); write('I O N I C S');
  gotoxy(27,9);write('Data Acquisition Module');
  gotoxy(21,12);write('Indian Institute of Technology Kanpur');
  gotoxy(32,14);write('1987');
  delay(2500);
end;                    {MENU_0}

procedure menu_1(var fn:name);

  {*  Function:    Menu for filename
                   loads the file name in fn (in upcase)
                   loads the file header and stores in hd
      Globals mod. : hd,fn                              *}

var done:boolean;
    i,j:integer;

begin
  clrscr;drawDbox(24,5,52,7);
  gotoxy(26,6);write('ION)CS Acquisition Module');
  gotoxy(18,15);
  write('Data File (or (CR) to quit) : ');
```


III.9

```
  repeat
    gotoxy(48,15);clreol;
    readln(con,fn);                                  { get file name       }
    if length(fn)=0 then exit;                       { and validate it     }
    done:=exist(fn);
    if done then
      begin write(^g);done:=false;end
    else
      writeopen(fn,done);
  until done;
  for i:= 1 to length(fn) do fn[i] := upcase(fn[i]); { convert file name to }
                                                     { uppercase letters   }
  gotoxy(18,17);write('File header : ');             { get file header     }
  i:=0;
  repeat
    i:=i+1;
    read(con,hd[i]);
  until hd[i]=chr(13);
  for j:= 1 to i do write(datfile,hd[j]);
  writeln(datfile);
end;                                          {MENU_1}

procedure menu_2( var choice:integer);

  {*  Function:   Menu for choosing options          *}

var ok:boolean;
begin
   first:=false;
   clrscr;drawDbox(4,2,32,4);
   gotoxy(6,3);write('ION)CS Acquisition Module');
   gotoxy(8,6);write('Data File : ');
   write(fn);
   gotoxy(8,8);write('OPTIONS');writeln(' (Frequency Control)');writeln;
   writeln('          0. Select new data file');
   writeln;
   writeln('          1. Spot mode');
   writeln('          2. Sweep mode');
   gotoxy(40,20);
   write('Your choice (0..2) : ');
   repeat
     gotoxy(61,20);clreol;
     {$I-} read(choice);{$I+}
     if (ioresult<>0) or (choice > 2) then
       write(^g);
   until (ioresult = 0) and (choice <= 2);
end;                                    {MENU_2}

procedure MENU_3(choice:integer);

  {*  Function:     Get parameters for data input
      Globals used: num                           *}

label 30,40;
var param:string[20];
```

```
    i,out:integer;
    num,start,stop,step:integer;
    range:array[1..7] of real;
    a:real;

begin;
   for i:=7 to 25 do
   begin
      gotoxy(1,i);clreol;
   end;
                              { get the values for frequency sweep }
   gotoxy(6,8);
   if choice=1 then
     begin
      write('No. of spot frequencies(kHz) : ');
      repeat
        gotoxy(37,8);clreol;
        {$I-} readln(num);{$I+}
        if ioresult<>0 then
          write(^g);
      until ioresult = 0;
      nofdata:=num;
      writeln;
      writeln('          No. : Spot frequencies(kHz) ');writeln;
      window(6,12,60,22);
      clrscr;
      for i:= 1 to num do
        begin
          gotoxy(1,i);
          write('              ',i:4,' :     ');
          repeat
            gotoxy(28,min(i,11));clreol;
            {$I-} readln(freq[i]);{$I+} out:=ioresult;
            if ((out<>0) or ((freq[i]<minfreq) or (freq[i]>maxfreq)))
                                                                    then
              write(^g);
          until (out = 0) and (freq[i]>=minfreq) and (freq[i]<=maxfreq);
        end;
        window(1,1,80,25);
     end
    else                      { get frequencies for auto sweep mode }
      begin
       nofdata:=0;
       range[1]:=0.01;range[2]:=0.1;range[3]:=1.0;
       range[4]:=10.0;range[5]:=100.0;range[6]:=1000.0;
       range[7]:=10000.0;
       gotoxy(57,7);write('OPTIONS :');
       for I:=1 to 7 do
         begin
           gotoxy(60,8+i);
           write(i,' : ',range[i]:8:2);
           gotoxy(73,8+i);write('kHz');
         end;
       30:gotoxy(10,8);
       clrtoxy(50,10);
```



```
       write('Start frequency : ');
       {$I-} read(start);{$I+}
       if (ioresult<>0) or (start>6) then
         begin
           write(^g);
           goto 30;
         end;
       if start<1 then exit;
       gotoxy(28,8);write(range[start]:8:2);
       gotoxy(38,8);write('kHz');
       freq[1]:=range[start];
       40:gotoxy(10,10);
       clrtoxy(50,10);
       write('Stop frequency  : ');
       {$I-} read(stop);{$I+}
       if (ioresult<>0) or (stop>7) then
         begin
           write(^g);
           goto 40;
         end;
       if (stop<=start) or (stop=0) then goto 30;
       gotoxy(28,10);write(range[stop]:8:2);
       gotoxy(38,10);write('kHz');
       gotoxy(10,12);
       clrtoxy(40,12);
       write('No. of steps/decade : ');
       repeat
         gotoxy(33,12);clrtoxy(40,12);
         {$I-} read(step);{$I+}
         if ioresult<>0 then
           write(^g);
       until ioresult = 0;
       if step<3 then goto 40;
       i:= 1;a:=exp(1/step*ln(10));
       while (freq[i]<=range[stop])
        begin
         i:=i+1;
         freq[i]:=freq[i-1]*a;
        end;
        nofdata:=i;
     end;
 end;

procedure  MENU_4;

{  Function  :  Menu for selecting the temperature control option   }

 var i,c:integer;
 begin
  for i:=7 to 25 do
  begin
     gotoxy(1,i);clreol;
  end;

  gotoxy(8,8);write('OPTIONS ');writeln('(Temperature Control)');writeln;
```

```
  writeln('          1. Manual');
  writeln('          2. Automatic');
  gotoxy(40,20);
  write('Your choice [0..2] : ');
  repeat
    gotoxy(61,20);clreol;
    {$I-} read(c);{$I+}
    if (ioresult<>0) or (c > 2) then
      write(^g);
  until (ioresult = 0) and (c <= 2);
  if c<=0 then begin back:=true;exit;end
  else back:=false;
  if c=1 then                                  { if auto mode then use DT2805 }
      dt2805:=false
  else
    dt2805:=true;
end;

procedure gpib_over(dev:integer);

{  Function  :  To set GPIB to initial status      }

 begin
  sbwrt(dev,CMDGRG);              { release the impedance analyser }
  ibloc(dev);                     { send it to local mode }
  window(1,1,80,25);
  gotoxy(55,23);write('Records written : ',count);
  window(17,15,66,22);
  clrscr;
  window(1,1,80,25)
 end;

procedure INITHP4192(var dev:integer);

{  Function  :  To initialize the HP4192      }

 begin
  brd0:=ibfind('GPIB0');          { find the GPIB card }
  ibasic(brd0);
  if error then
    begin
      finderror;
      exit
    end;                          { get the identifier for the impedance analyser }
  dev:=ibfind(devglb);            { named as HP4192 in the global name devglb     }
  if error then
    begin
      finderror;
      exit
    end;
  ibclr(dev);
  if error then
    begin
      finderror;
      exit
```



```
    end;
  sbwrt(dev,CMDGLB);
  if error then finderror;
 end;

procedure GETHPDATA(i:integer;tempr:real;dev:integer);

{  Function  :  Get the data from HP4192 and output it on the
                screen and the disk file with file name in fn   }
 begin
   sbwrt(dev,'FR'+r_ASCII(FREQ[I])+'EN'+'EX');
   writehp4192data(tempr,dev);
 end;

procedure t_manual;

{  Function  :  Control the flow of the program in case of
                manual temperature control mode                 }

 var i,impal:integer;
 begin

  repeat                  { loop until no more readings to be taken }
    window(1,1,80,23);
    clrscr;
    gotoxy(27,11);write('     Press a key when ready');
    repeat until keypressed;
    inithp4192(IMPAL);
    if giberr then exit;
    gotoxy(27,11);clreol;
    write('     Temperature[°K] = ');
    repeat
      gotoxy(50,11);clreol;
      {$I-} read(tempr);{$I+}
      if (ioresult<>0) or (tempr=0) then
          write(^g);
    until ((ioresult = 0) and (tempr <> 0));
    gotoxy(17,13);
    write(DATALIST);
    drawbox(16,14,67,23);
    gotoxy(17,14);
    for i:= 1 to nofdata do
     gethpdata(i,tempr,IMPAL);
    gpib_over(IMPAL);
    gotoxy(32,11);write('   ');textcolor(black+blink); textbackground(white);
    write(' Repeat(Y/N) ');textbackground(black);write('         ');
    textcolor(white);
    gotoxy(1,21);
    repeat
      read(kbd,cha);
      if not(cha in ['n','N','y','Y']) then
          write(^g);
    until cha in ['n','N','y','Y'];
  until cha in ['n','N'];                { end of the loop }
  window(1,1,80,25);                     { set the window to full screen  }
```

```
  sbwrt(IMPAL,CMDORG);              { release the impedance analyser }
  ibloc(IMPAL);                     { send it to local mode          }
end;

{################################## t_auto ##################################}

procedure t_auto;

{*  Function  :  To control the flow of the program in case of the
                 auto temperature control mode                          *}

  label 200,210,300,330;

  type device = string[3];
       name   = string[20];
  var  tkelvin:array [1..300] of real;
       ref1,ref,fglb:real;
       rate1,rate2,hold1:real;
       start,stop,step,prvsp,prvstp,prvsta,moderr,hold,dumcnt:real;
       ch,noftemp:integer;
       hglb,mglb,sglb,t_n_minus1:real;
       prv1,prvcnt,num:integer;
       cool,first:boolean;

  procedure timer(var hr,min,sec,frac:integer);

  {*  Function :  To interrogate the system real time clock
                  This function returns the time in HR:MIN:SEC:SEC/100 in
                  hr,mion,sec,frac                                      *}

    type
        dosreg = record
                   AX,BX,CX,DX,BP,SI,DI,DE,DS,FLAGS:integer;
                 end;
    var
        reg:dosreg;
    begin
      with reg do
        begin
          AX:=$2C00;                { call MS-DOS function $2C       }
          MSDOS(reg);               { to interrogate the system clock }
          hr:=hi(CX);
          min:=lo(CX);
          sec:=hi(DX);
          frac:=lo(DX);
        end;
    end;

  function Seconds(hr,min,sec:real):real;

  {*  Function  :  Converts the time (hr:min:sec) into seconds  *}

    begin
      seconds:=(hr*60 + min)*60 + sec
    end;
```



```
  procedure Gettime;

  {*  Function  :  Determine the time elapsed since the start of
                   the experiment.  The reference time is stored
                   in variable REF. Every time it is called, it stores the
                   time elapsed in HGLB:MGLB:SGLB which are globals for
                   t_auto.
      Globals used : None
      Globals mod. : ref,hglb,mglb,sglb                                 *}

    var
       rhr,rmin,rsec,rfrac:integer;
       tim:real;
    begin
      timer(rhr,rmin,rsec,rfrac);
      tim:=seconds(rhr,rmin,rsec) + rfrac/100;
      if tim < t_n_minus1 then ref:=-(ref1);
      t_n_minus1:=tim;
      ref1:=tim - ref;
      hglb:=trunc(ref1/3600);
      ref1:=ref1-hglb*3600;
      mglb:=trunc(ref1/60);
      ref1:=ref1-mglb*60;
      sglb:=trunc(ref1);
      fglb:=ref1-sglb;
    end;

  procedure DISPLAYTIME;

  {*  Function  :  Display the time elapsed on the screen.
                   Writing to the screen buffer
      Globals used : hglb,sglb,mglb
      Globals mod. : None                                               *}

    begin
      gotoxy(66,3);write(hglb:2:0);
      gotoxy(69,3);write(mglb:2:0);
      gotoxy(72,3);write(sglb:2:0);
    end;

  procedure menu_22( var choice:integer);

    {*  Function:   Menu for choosing options for furnace control      *}

  var ok:boolean;
      i:integer;
  begin
     first:=false;
     for i:=7 to 25 do
     begin
       gotoxy(1,i);clreol;
     end;

     gotoxy(8,8);write('Furnace Control :');
     gotoxy(8,10);writeln('OPTIONS');writeln;
```

```
        gotoxy(34,8);clreol;
        {$I-} read(start);{$I+}out:=ioresult;
        if out<>0 then
        write(^g);
      until out = 0;
    if start<=0 then exit;
    tkelvin[1]:=start;
    gotoxy(10,10);
    write('Stop Temperature(°K)  : ');
    40:repeat
        gotoxy(34,10);clreol;
        {$I-} read(stop);{$I+}out:=ioresult;
        if out<>0 then
          write(^g);
      until out = 0;
    if stop<=0 then goto 30;
    gotoxy(10,12);
    write('Step size             : ');
    repeat
      gotoxy(34,12);clreol;
      {$I-} read(step);{$I+} out:=ioresult;
      if out<>0 then
        write(^g);
    until out = 0;
    if step<=0 then goto 40;
    getrates;
    if initrate <=0 then exit;
    i:=1;
    if stop < start then
      begin
        step:=-step;
        while (tkelvin[i]>stop)
          begin
            i:=i+1;
            tkelvin[i]:=tkelvin[i-1]+step;
          end;
      end
    else
      begin
        while (tkelvin[i]<stop)
          begin
            i:=i+1;
            tkelvin[i]:=tkelvin[i-1]+step;
          end
      end;
    noftemp:=i;
  end;
end;

procedure ScreenOut;

{  Function  :  Make the final screen on text page zero  }

  begin
    clrscr;
```



```
    drawdbox(1,2,34,4);gotoxy(3,3);write('IONICS Data Acquisition Module');
    drawbox(64,2,75,4);
    gotoxy(59,3);write('Time');
    gotoxy(68,3);write(':');gotoxy(71,3);write(':');
    drawbox(10,8,22,10);gotoxy(19,9);write('°K');
    gotoxy(12,7);write('Set Point');
    drawbox(59,8,71,10);gotoxy(68,9);write('°K');
    gotoxy(62,7);write('Actual ');
    gotoxy(37,7);write(' Power');drawbox(36,8,44,10);
    gotoxy(42,9);write('%');
    drawbox(16,14,67,23);gotoxy(39,14);write(' Data ');
    gotoxy(10,24);textcolor(black);textbackground(white);
    write('Ctrl-F1');gotoxy(24,24);write('Ctrl-F10');
    textbackground(black);textcolor(white);
    gotoxy(5,24);write('Exit');gotoxy(19,24);write('Edit');
    gotoxy(55,24);write('Records written : ',count);
  end;

procedure Getsetpt(index,tport:integer;var d_C,setpoint:real);

{  Function  :  To the current set point as determined by the
                heating rate specified in the auto temp. control
                mode                                              }

  var rate:real;
  begin
    if index=1 then rate:=initrate
               else rate:=frate;
    gat(1,tport,d_C);
    if d_C > tkelvin[index] - 273 then rate:=-rate;
    if ((tkelvin[index]-273) > (setpoint -abs(rate)))
       and ((tkelvin[index]-273) < (setpoint +abs(rate))) then
      setpoint:=tkelvin[index]-273
    else
      begin
        e[1]:=e[1]+rate; e[2]:=e[2]+rate; e[3]:=e[3]+rate;
        setpoint:= setpoint + rate;
      end;
  end;

procedure PID(error:real;var merror:real);

{  Function  :  To set up the power to be delivered to the furnace
                in each control loop, depending on the heating rate,
                current actual temp. by using the PID algorithm   }

  var pp:real;
  begin
    e[3]:=error;
    merror:=ME+KC*((e[3]-e[2])+(e[3]/tint)+tdiff*(e[3]-2*e[2]+e[1]));
    if merror<0 then merror:=0;
    if merror>nofsteps then merror:=nofsteps;
    ME:= merror;e[1]:=e[2];e[2]:=e[3];
    merror:=merror/nofsteps;                  { translate the modified }
    pp:=merror * 100;                         { error to % power       }
```

```
    if pp > 100 then pp:=100;
    if pp < 0 then pp:=0;
    gotoxy(38,9);write(round(pp):3);
  end;

procedure ctrl_temp(port:byte);

{  Function  : To control the temp. by delivering the power calculated by
               PID
   Globals used : dumcnt,moderr,prvcnt
   Globals mod. : prvcnt,dumcnt                                          }

   begin
     dumcnt:=dumcnt + moderr;
     if prvcnt=trunc(dumcnt) then
       begin
         xodv (port,OFF_CTRL_BYTE,OFF_CTRL_BYTE);
       end
     else
       begin
         xodv(port,ON_CTRL_BYTE,ON_CTRL_BYTE);
         dumcnt:=frac(dumcnt);
         prvcnt:=trunc(dumcnt);
       end
   end;

procedure DRIVEF(indx:integer;port:byte;var setpoint:real;
                 var ready:boolean);

{  Function  :  Control the furnace temperature
   Globals used : t_port,tkelvin[i]
   Globals mod. : None
   Routines used : ctrl_temp                                             }

  var  deg_C:real;

  begin
    ctrl_temp(port);xmt(1,t_port,deg_C);
    ready:=false;
    gotoxy(60,9);write(deg_C+273:7:1);
    if abs(deg_C+273-tkelvin[indx])<1 then
    begin
      ready:=true;
      setpoint:=tkelvin[indx]-273
    end;
  end;

procedure InitPID;

{  Function  :  To initialize the parameters of PID
   Globals used : kc_rel,nofstep,ti_rel,td_rel,f_port,t_port,e[i],me
   Globals mod. : kc_rel,nofstep,ti_rel,td_rel,f_port,t_port,e[i],me  }

  var dummy:integer;
      deg_cent,d:real;
```



```
  begin
    KC:=kc_rel*nofsteps; tint:=ti_rel/nofsteps; tdiff:=td_rel/nofsteps;
    XRD(dummy);xefo(f_port);xodv(f_port,0,0);xmt(1,t_port,deg_Cent);
    d:=0; me:=0;
    e[1]:=d;e[2]:=d;e[3]:=d;
  end;

procedure init_time;

{  Function  :  To initialize the reference time
   Globals used : ref
   Globals mod. : ref
   Routines used : timer                                                 }

  var rhr,rmin,rsec,rfrac:integer;
  begin
    timer(rhr,rmin,rsec,rfrac);
    ref:=seconds(rhr,rmin,rsec) + rfrac/100;
    t_n_minus1:=ref;
  end;

procedure FURTREND(i:integer);

{  Function  :  To determine the trend of
                the furnace (heating or cooling)                         }

  var deg_Cent:real;
  begin
    gotoxy(11,9);write(tkelvin[i]:7:1);xmt(1,t_port,deg_Cent);
    if (deg_Cent+273) <= tkelvin[i] then
      begin
        gotoxy(36,19);
        write('....heating');
        cool:=false;
      end
    else
      begin
        gotoxy(36,19);
        write('....cooling');
        cool:=true;
      end;
  end;

procedure MONITERFUR(i,port:integer;sp:real;tsecref:real;
                     holding_tim:real;holding:boolean;
                     var rdy:boolean);

{  Function  :  To monitor the furnace every second.
                The control returns from this if interrupted
                or furnace ready or the control loop time (nofstep)
                is over.  The var. HOLDING is true if called from the
                holding loop and the referance time when the hold began
                is in HOLDING_TIM.
   Globals used : nofsteps,hglb.mglb,sglb
   Globals mod. : atn
```

```
    Routines used : drivef                                          *)

  var tim_now,ti:real;

    begin
      tim_now:=tsecref;ti:=tsecref;
      while (not(rdy)) and (tim_now - tsecref < nofsteps) do
      begin
        drivef(i,port,sp,rdy);
        if holding then
          begin
            rdy:=false;
            if ((tim_now - holding_tim) >= hold) then exit;
          end;
        repeat
          gettime;
          tim_now:=seconds(hglb,mglb,sglb);
          if keypressed then
            begin
              atn:=true;tsecref:=tim_now;exit;
            end
          else atn:=false;
        until ((tim_now-ti) >= 1);
        displaytime;ti:=tim_now;
      end;
    end;

  function change:boolean;

  (*  Function  :  To test if changes have been made in the edit mode of the
                   auto temp. control
      Globals used : start,prvsta,step,prvstp
      Globals mod. : None
      Routines used : None                                          *)

    begin
      if tkelvin[i] <> prvsta then change:=true
      else
        if step <> prvstp then change:=true
        else change:=false;
    end;

  procedure MASTER(port:byte);

  (*  Function  :  The master control routine of the auto control mode
      Globals used : dumcnt,edit,prvsp,glberr,nofteap,moderr,atn,hold,
                     tkelvin[i],t_port,nofdata,hglb,mglb,sglb,me
      Globals mod. : dumcnt,edit,prvsp,atn
      Routines used : inithp4192,furtrend,gettime,seconds,getstpt,PID,
                      moniterfur,drivef,displaytime,ctrl_temp,gethpdata,
                      gpibover                                      *)

    label 1,2,10;
    var
        ar,yr,h,s,f,impal,dummy,i,j:integer;
```



```
        d,tsec,tsecr,templ,sp,deg_Cent,deg_C:real;
        rdy:boolean;

    function Interrupt(tim,tref:real;holding:boolean):boolean;

    (* Function      : To process an keyboard activity during the run.
                       If the keyboard input is an interrupt (Ctrl-F1 or
                       Ctrl-F10) then returns true else false.  This is for
                       use by MASTER routine only.                  *)

    var cch:char;
    begin
      interrupt:=false;
      read(kbd,cch);
      if (cch = #27) and (keypressed) then
        begin
          read(kbd,cch);
          case cch of
            #94:begin interrupt:=true;exit;end;        {if Ctrl-F10 then end}
            #103:begin                                 {if Ctrl-F1 then edit}
                   interrupt:=true;
                   edit:=true;prvl:=1;prvsp:=sp;exit;
                 end
          end;{of case}
        end;
      if holding then                          {if not an interrupt then }
        if not(tim - tref >=hold) then         {then check the loop      }
          begin                                {termination condition for}
            atn:=false;interrupt:=false;       {the calling loop         }
          end
      else
        if not(rdy) then begin atn:=false;interrupt:=false;end
    end;

    begin
      dumcnt:=0;prvcnt:=0;
      if not(edit) then                          { if parameters edit   }
        begin                                    { interrupt used and   }
          xmt(1,t_port,deg_Cent);sp:=deg_Cent;   { parameters changed   }
        end                                      { then initialize      }
      else
        if change then
          begin
            xmt(1,t_port,deg_Cent);sp:=deg_Cent;initpid;
          end
        else sp:=prvsp;
      inithp4192(impal);              { check if impedance measuring }
      if glberr then                  { device present               }
      begin ibloc(impal);exit;
      end;
      ibloc(impal);                   { if yes the send it to local  }

    { THE MAIN CONTROL LOOP }

      for i:=1 to nofteap do
```

```
begin
  if not(change) then I:=prvI;edit:=false;
  rdy:=false;
  furtrend(i);

  { LOOP TILL FURNACE READY OR INTERRUPTED }
  {##########################################################}
  repeat
    getstpt(i,t_port,deg_Cent,sp);d:=sp-deg_Cent;
    PID(d,moderr);
    gettime;tsec:=seconds(hglb,mglb,sglb);
    I:MONITERFUR(i,port,sp,tsec,0,false,rdy);
  until rdy or atn;
  {##########################################################}

  if atn and not(rdy) then                    {if interrupted then}
    if interrupt(tsec,tsec,false) then exit   {goto the interrupt }
    else goto 1;                              {handler for approp-}
                                              {riate action.       }
  tsecr:=tsec;
  gotoxy(36,19);write('....holding');
  sp:=t[elvin(i) - 273;
  rdy:=false; { ---------------------- only for the purpose of the }
                                  { holding loop                    }
  { HOLDING LOOP FOR hold NUMBER OF MINUTES }
  {##########################################################}
  repeat
    xat(1,t_port,deg_Cent);d:=sp-deg_Cent;
    PID(d,moderr);
    gettime;tsec:=seconds(hglb,mglb,sglb);
    2:MONITERFUR(i,port,sp,tsec,tsecr,true,rdy);
  until (tsec - tsecr )= hold) or atn;
  {##########################################################}

  if atn and not(tsec - tsecr )= hold) then   {if interrupted then}
    if interrupt(tsec,tsec,false) then exit   {goto the interrupt }
    else goto 2;                              {handler for approp-}
                                              {riate action.       }
  gotoxy(17,13);
  inithp4192(impal);
  if glberr then
  begin ibloc(impal);exit;
  end;
  write(DATALIST);
  gotoxy(17,14);
  xat(1,t_port,temp1);
  if cool then temp1:=-(temp1+273)
  else temp1:=temp1+273;
  gettime;
  displaytime;
  gotoxy(18,14);beep;

  { LOOP TO READ AND WRITE DATA (to screen and datafile) }
  {##########################################################}
  for j:= 1 to nofdata do
```



```
    begin
      ctrl_temp(port);
      gethpdata(j,temp1,impal);gettime;
      if glberr then
      begin ibloc(impal);exit;
      end;
      if (me-moderr))>nofsteps then moderr:=me;
    end;
  {##########################################################}

    gpib_over(impal);
    gotoxy(17,13);clreol;flush(datfile);
  end;

{ END OF MAIN CONTROL LOOP }

  end;

begin        { t_auto main }
  back:=false;prvsta:=0;prvstp:=0;step:=0;
  xsetupdt;atn:=false;edit:=false;
  initpid;
  300:menu_2(ch);
  if ch<=0 then begin back:=true;xrd(ch);exit;end
  else back:=false;
  330:menu_3(ch);
  if (nofstep(=0) or (initrate (= 0) then goto 300;
  screenout;if not(edit) then init_time;
  master(f_port);
  if edit then
    begin
      atn:=false;
      prvsta:=tkelvin[1];prvstp:=step;
      xodv(f_port,0,0);goto 300;
    end;
  gettime;display_time;
end;         { t_auto }

{################################ t_auto ##################################}

begin                                  { MAIN }
  count:=0;
  glberr:=false;
  first:=true;
  menu_0;
  100:menu_1(fn);
  if length(fn)=0 then goto 200;
  300:menu_2(ch);
 if ch<=0 then goto 100;
 350:menu_3(ch);
 if nofdata(=0 then goto 300;
 400:menu_4;
 if back then goto 350;
 if dt2005=true then t_auto
               else t_manual;
```

```
  if back then goto 400;
  200:if first then goto 210;
  close(datfile);
  if (count=0) then erase(datfile);
  210:if not(glberr) then clrscr;
  gotoxy(32,12);write('Records  written : ',count);
  if chk > 127 then begin assign(datfile,'ionics.com');execute(datfile);end
end.                                                    { MAIN }
```

```
{*************************************************************************}
{*                                                                       *}
{*                  Driver for DT2805 ADC/DAC card                       *}
{*                  Programmer:Sanjay Bhatnagar                          *}
{*                                                                       *}
{*************************************************************************}

type datarray=array[1..16] of integer;
const base_address=$2EC;                    {base address of DT2805}
var data_in,data_out,command,status:integer;

function XERROR:boolean;                    {error bit}

{* This checks the status byte of the DT2805 card.  If an error
   occured in the last operation with the card it set to TRUE otherwise
   it returns a value FALSE                                            *}

  var t:byte;
  begin
    t:=port[status];
    xerror:=((t and $80)=$80);              {Check status 7 bit}
  end;

function XCOMMAND:boolean;                  {command bit}

{*  This function checks the status byte to see if the card has
       received the last byte as command or data                       *}

  var t:byte;
  begin
    t:=port[status];
    xcommand:=((t and $08)=$08);            {Check status 3 bit}
  end;

function XREADY:boolean;                    {ready bit}

{* This function checks the status byte to see if the card has
   completed the last operation.  If the last operation is over then
   it returns a value TRUE otherwise FALSE                             *}

  var t:byte;
  begin
    t:=port[status];
    xready:=((t and $04)=$04);              {Check status 2 bit}
  end;

function X_DATA_IN_EMPTY:boolean;           {data in full bit}

{* This function checks the status byte to find if the input register
   is empty.  If it is empty, the function returns a value TRUE otherwise
   it returns a value FALSE                                            *}

  var t:byte;
  begin
    t:=port[status];
```



```
    x_data_in_empty:=not((t and $02)=$02);  {Check status 1 bit}
  end;

function X_DATA_OUT_READY:boolean;          {data out redy bit}

{* This function checks the status byte to find if the output register
   is ready to output data.  If it is empty, the function returns a value
   TRUE otherwise it returns a value FALSE                             *}

  var t:byte;
  begin
    t:=port[status];
    x_data_out_ready:=((t and $01)=$01);    {Check status 0 bit}
  end;

function READ_BYTE:byte;

{* This function returns the byte in the data out register             *}

  begin
    repeat until x_data_out_ready;
    READ_BYTE:=port[data_out];
  end;

procedure XSETUPDT;                         {initialisation of addresses}

{* This routine sets up the addresses of the various ports to be used
   by the DT2805 card.  This routine must be the first call in the sequence
   to start communications with the DT2805 card.  It needs to be called
   only once.                                                          *}

  begin
    data_in:=base_address;
    data_out:=base_address;
    command:=base_address+1;
    status:=base_address+1;
  end;                                      {setupdt}

procedure XSTOP;

{* This procedure stops the current continous operation in progress    *}

  var t:byte;
  begin
    port[command]:=$0F;
    t:=port[data_out];
  end;

procedure XRD(var t:integer);

{* This routine initializes the DT2805 card for further operations.  This
   normally should be the called immediately after a call to the set up
   routine X_SET_UP_DT                                                 *}

  var dummy:byte;
```

```
    begin
      xstop;
      dummy:=port[data_out];
      repeat until xready;
      if xerror then
      begin
        port[command]:=$01;
        repeat until xready;
      end;
      port[command]:=$00;
      repeat until x_data_out_ready;
      t:=port[data_out];
      repeat until x_data_in_empty;
    end;

procedure XODV(portno:byte;mask,val:integer);

{* This routine outputs a digital value to the a selected port with a given
   mask.  The selected port comes in portno as a byte, the mask to be used
   for the output and the value to be outputed comes in the variables mask and
   val.                                                                    *}

  var t:byte;
      ti:integer;
  begin
    repeat until xready;                          {write Digital output immediate}
    port[command]:=$07;
    repeat until x_data_in_empty;
    port[data_in]:=portno;
    ti:=mask and val;                             {perform masking}
    repeat until x_data_in_empty;
    t:=ti;
    port[data_in]:=t;                             {write masked value to port}
    if portno=$02 then                            {write the high byte}
      begin
        t:=ti div 256;
        repeat until x_data_in_empty;
        port[data_in]:=t;
      end;
    repeat until x_data_in_empty;
  end;

  procedure XEFO(portno:byte);

  {* This routine enables the port chosen in the variable portno for output *}

    var t:byte;
    begin
      repeat until xready;
      port[command]:=$05;                         {EFO command}
      repeat until x_data_in_empty;
      port[data_in]:=portno;
      repeat until x_data_in_empty;
      xodv(portno,0,0);
    end;
```



```
procedure XEFI(portno:byte);

{* This routine enables the port chosen in the variable portno for input *}

  var t:byte;
  begin
    repeat until xready;
    port[command]:=$04;                           {EFI command}
    repeat until x_data_in_empty;
    port[data_in]:=portno;
    repeat until x_data_in_empty;
  end;

procedure XSA(source,channel_st,channel_fin,gain:integer);

{* This routine sets up the various parameters of the ADC.  The variable
   source is not used, but is intended to be used for setting up the kind
   of timming source required.  The varibles channel_st,channel_fin gets
   the start and the finishing channel numbers to be scanned and the
   variable gain gets one of the possible gains to be used for the channels
   used.                                                                   *}

  var t:byte;
  begin
    repeat until xready;
    port[command]:=$00;
    repeat until x_data_in_empty;
    case gain of                                  {set up the gain}
      500: port[data_in]:=$03;
      100: port[data_in]:=$02;
       10: port[data_in]:=$01;
        1: port[data_in]:=$00
    end;
    repeat until x_data_in_empty;
    t:=channel_st;
    port[data_in]:=t;
    repeat until x_data_in_empty;
    t:=channel_fin;
    port[data_in]:=t;
    repeat until x_data_in_empty;
    port[data_in]:=$03;                           {dummy no. of conversions}
    repeat until x_data_in_empty;
    port[data_in]:=$00;                           {high byte}
    repeat until x_data_in_empty;
  end;

procedure XAS(no_of_values:integer;var values:dataarray);

{* This routine reades the ADC continously. The variable no_of_values stores
   the number of convertions required, and gives the read values in the array
   variable values. This variable is 16 locations (of integer type) long. If
   no_of values is more than 16 change this length in the global decleration
   of dataarray accordingly.                                               *}
```

```
  ar l,h:byte;
     i:integer;
  egin
   repeat until xready;
   port[command]:=$2E;                          (Read A/D continous)
   for i:=1 to no_of_values do

       i:=read_byte;
       h:=read_byte;
       values[i]:=l+256*h;
     end;
    xstop;                                      (stop conversions)
  end;

procedure sort(var nums:datarray;num:integer);

(* This routine does the quick sort of the values in the variable nums and
   the sorted values are put back in the same variable. The sorted values are
   in the ascending order.                                                  *)

   var temp1:integer;
       j,m,i,j,pass,gap:integer;
   procedure flipflop;
     begin
       temp1:=nums[j];
       nums[j]:=nums[j+gap];
       nums[j+gap]:=temp1;
     end;
   begin
     for k:=1 to num do
     begin
       gap:=num div 2;
       while gap>0 do
         begin
           for i:=(gap) to num do
             begin
               j:=i-gap;
               while j>0 do
                 begin
                   if nums[j]>nums[j+gap] then
                     begin
                       flipflop;
                       j:=j-gap;
                     end
                   else j:=0;
                 end;
             end;
           gap:=gap div 2;
         end;
     end;
  end;
```



```
procedure XMT(tctype,channel:integer;var deg_C:real);

(* This function reads the channel indicated by the variable channel and
   gives the equivalent temperature in the variable deg_C.  The variable
   tctype is not used but is intended to be used to specify the type of
   thermocouple used to measure the temperatue. Currently only type K
   Chromel-Alumel  is supported.  The routine has the capabilities
   to set up the required gain automatically  if the requirment lies in
   the range of >=100 and (= 500. For more ranges, the setgain routine will
   have to be modified.                                                    *)

label 100;
var cdata:array[1..2,1..2] of real;
    tcv,rt,sumav:real;
    index:integer;
    temp:datarray;

  procedure setgain(chn:integer);

  (* This routine is used by XMT to auto set the gain of the channel number
     chn. The data required for each gain is in the global variable cdata.
     The logica used is to start with the minimum gain and take a reading.
     Keep switching to the next higher gain till the reading taken is (= to
     80% of the full scale reading of the current gain.                    *)

    begin
      xsa(0,chn,chn,100);
      index:=2;
      xas(1,temp);
      tcv:=cdata[2,index]+cdata[1,index]*temp[1];
      if tcv>abs(0.8*(4095*cdata[1,index]+cdata[2,index])) then
        begin
          xsa(0,chn,chn,500);
          index:=1
        end;
    end;

  procedure xmt1;

   (* Routine used by XMT to get the average value of 16 reading after
      filtering. The median 8 of the sorted values are used to get the
      average value.                                                       *)

  var temp:datarray;
      sum,i:integer;
    begin
      xas(16,temp);
      sort(temp,16);
      sum:=0;
      for i:=5 to 12 do
          sum:=sum+temp[i];
      sumav:=sum/8.0;
    end;
```

```
begin
  cdata[1,1]:=9.7656E-03;  cdata[2,1]:=-20.00;          {gain 500}
  cdata[1,2]:=48.828E-03;  cdata[2,2]:=-100.0;          {gain 100}
  setgain(0);
  xmtl;
  rt:=(cdata[2,index]+cdata[1,index]*sumav)*2.0;        {0.5mV per °C}
  if channel=0 then
    begin
      deg_C:=rt;
      goto 100;
    end;
  setgain(channel);
  xmtl;
  tcv:=cdata[2,index]+cdata[1,index]*sumav;                    {mV output}
  deg_C:=tcv/0.041269 + rt;                                 {1000°C = 41.269 mV}
  100:
end;
```

```
{*****************************************************************************}
{*                                                                           *}
{*                  SUPERIONIC CONDUCTIVITY DATA ACQUISITION                 *}
{*                                                                           *}
{*                          Data Generation Module                           *}
{*                             Sanjay Bhatnagar                              *}
{*                              September 1987                               *}
{*                                                                           *}
{*****************************************************************************}

PROGRAM TEXTEDITOR;

{* The program is line editor and provides a data entry option specifically *}
{* for the data formats of the IONICS system. The program loads the file to *}
{* be edited in the memory  line by line and maintains a double linked list *}
{* of the lines. The details of the lines is stored in a record, and the    *}
{* linepointer holds the pointer to this record. It has four fields: NUM,   *}
{* TXT,NEXT,PRIOR. NUM holds the line number, TXT holds the text of that    *}
{* line, NEXT is a pointer which points to the record of the next line, and *}
{* PRIOR points to the record of the previous line.The routines insertline, *}
{* dl delete, inserts and deletes lines in the list and maintain the link.  *}
{* INSERT_IN_LINE and DELETE_FROM_LINE does the operation on the text of a  *}
{* particular line and does not temper with links. The variables START and  *}
{* LAST holds the pointers to the first and the last lines of the file.     *}
{* The listing of the directory contents and changing the current directory *}
{* is done by the routine DIRECTORY. This has routines to set the DTA (DISK *}
{* TRANSFER AREA), search the directory for the mask given and to set up the*}
{* proper mask. This is done by calling the DOS functions 1Ah,4Eh,4Fh       *}
{* the interrupt 21h.                                                      *}

 const Linmsg = 'Line number: ';
 type
   str255 = string[255];
   linepointer =^line;
   line = record
          txt:str255;           {text of the line in a file  }
          num:real;             {line number                 }
          next:linepointer;     {pointer to the next  line   }
          prior:linepointer;    {pointer to the previous line}
   end;
   filetype = text;
 var
   cnt:byte absolute Cseg:$80;    {variable to find the method of loading the
                                   module (see the IONICS manual) and execute
                                   command of Turbo Pascal from the technical
                                   reference manual)                         }
   text : filetype;
   check:byte;
   start,last,prv,prvsav:linepointer;
   sts,newfile,freq,ok,done:boolean;
   ch:char;
   prvline:real;
   op:integer;
   param:string[15];
   dir_ref,conststr,hd,fn:str255;
```



```
procedure drawbox(x1,y1,x2,y2:integer);

{*  Function  : To draw box with upper left corner at (X1,Y1)
                and lower right corner at (X2,Y2)
    Globals used : None
    Globals mod. : None
    Routines used : None                                                   *}

  var index:integer;
    begin
     gotoxy(x1,y1);write('┌');
     for index:= 1 to (x2-x1-1) do write('─');
     write('┐');
     for index:= (y1+1) to (y2-1) do
       begin
         gotoxy(x1,index);write('│');
         gotoxy(x2,index);write('│');
       end;
     gotoxy(x1,y2);write('└');
    for index:= 1 to (x2-x1-1) do write('─');
    write('┘');
    gotoxy(x1,y1);
   end;

procedure drawDbox(x1,y1,x2,y2:integer);

{*  Function  : To draw double box with upper left corner at
                (X1,Y1) and lower right corner at (X2,Y2)
    Globals used : None
    Globals mod. : None
    Routines used : None                                                   *}

  var index:integer;
  begin
    gotoxy(x1,y1);write('╔');
    for index:= 1 to (x2-x1-1) do write('═');
    write('╗');
    for index:= (y1+1) to (y2-1) do
      begin
        gotoxy(x1,index);write('║');
        gotoxy(x2,index);write('║');
      end;
    gotoxy(x1,y2);write('╚');
    for index:= 1 to (x2-x1-1) do write('═');
    write('╝');
    gotoxy(x1,y1);
  end;
```

```
function Find(lnum:real):linepointer;

{ Function     : To find the pointer to the line with line number lnum
  Globals used : start
  Globals mod. : None
  Routines used: None                                                  }

 var i:linepointer;
 begin
   i:=start;
   find:=nil;
   while (i<>nil) do
     begin
       if lnum = i^.num then find:=i;
       i:=i^.next;
     end;
end;

procedure Patchup(lnum,incr:real);

{ Function     : To renumber line numbers from lnum with increment incr
  Globals used : None
  Globals Mod. : None
  Routines used: Find                                                  }

 var i:linepointer;
 begin
   i:=find(lnum);
   while (i<>nil) do
     begin
       i^.num:=i^.num+incr;
       i:=i^.next;
     end;
 end;

function dl_delete(start:linepointer;key:real):linepointer;

{ Function     : To delete a line with number KEY from the list
  Globals used : start
  Globals mod. : None
  Routines used: patchup                                               }

 var temp,temp2:linepointer;
     done:boolean;
 begin
   if start^.num = key then          {if the line is the first line of}
     begin                           {the list then remove it         }
       dl_delete:=start^.next;
       if temp^.next <> nil then
         begin
           temp:=start^.next;
           temp^.prior:=nil;
         end;
         dispose(start);
     end
```



```
   else
     begin                         {if the line is not the first line    }
       done:=false;                {then remove it and connect the broken}
       temp:=start^.next;          {links of the list                    }
       temp2:=start;
       while (temp <> nil) and (not done) do
         begin
           if temp^.num = key then
             begin
               temp2^.next:=temp^.next;
               if temp^.next <> nil then
                 temp^.next^.prior := temp2;
               done:=true;
               last:=temp^.prior;
               dispose(temp);        { release the memory }
             end
           else
             begin
               temp2:=temp;
               temp:=temp^.next;
             end;
         end;
       dl_delete:=start;
       if not done then begin writeln('Line not found');writeln;end
       else patchup(key+1,-100);
     end;
   end;

procedure insertline;

{ Function     : To insert a line at a given position in the list
  Globals used : start
  Globals mod. : None
  Routines used : find,patchup                                         }

  var old,i,into:linepointer;
      num,ref:real;
      x,y:integer;
  begin
    num:=-1;
    write(linmsg);x:=wherex;y:=wherey;
    repeat                                  {get the position of insertion}
      gotoxy(x,y);clreol;
      {$I-} read(num); {$I+}
    until ioresult = 0;
    writeln;
    if num < 0 then begin writeln;exit;end;
    i:=find(num);ref:=num;
    if i = nil then
      begin
        if num < start^.num then            {if the position is before the}
          begin                             {first line then insert at the}
            i:=start;                       {beginning                    }
            old:=start;
            write(num + 1:6:0,': ');
```

```
            num:= num + 1;
            new(info);readln(con,info^.txt);
            info^.num:=num;
            info^.prior:=nil;
            info^.next:=start;start^.prior:=info;
            start:=info;

            repeat                                  {remain in the insert mode  }
              old:=info;                            {till the end of the section}
              write(num + 1:6:0,': ');              {is indicated               }
              num:=num + 1;
              new(info);readln(con,info^.txt);
              info^.num:=num;
              info^.prior:=old;old^.next:=info;
              info^.next:=i;
            until ((info^.txt[0] = #1) and (info^.txt[1] in ['e','E']));

            old^.next:=i;
            dispose(info);
            exit;
          end
        end
      else
        begin                                       {if the position is not}
          repeat                                    {the first line then   }
            num:=num + 1;                           {insert the line and   }
            write(num:6:0,': ');                    {maintain the link     }
            new(info);readln(info^.txt);
            info^.next:=i^.next;i^.next:=info;
            info^.prior:=i;info^.num:=i^.num +1;i:=info;
          until (info^.txt[1] in ['e','E']);
          info^.prior^.next:=info^.next;
        end;
      patchup(ref+1,100);
    end;

  procedure remove(var deleted:boolean);

  {* Function     : To remove a line from the list
     Globals used : start
     Globals mod. : None
     Routines used: dl_delete                    *}

    var num,to_num,temp:real;
        x,y:integer;
        sptr,fptr:linepointer;
    begin
      deleted:=false;
      write(linmsg);x:=wherex;y:=wherey;
      num:=-1;to_num:=0;
      repeat
        gotoxy(x,y);clreol;                 {get the line number to be deleted}
        {$I-} read(num,to_num); {$I+}
      until ioresult = 0;
      writeln;
```



```
      sptr:=find(num);
      if sptr <> nil then
        begin
          if to_num = 0 then
            fptr:=sptr^.next
          else
            begin
              fptr:=find(to_num);
              if fptr = nil then
                begin
                  writeln('End line not found');
                  exit;
                end
              else fptr:=fptr^.next;
            end
        end
      else begin writeln('First line not found');exit;end;
      deleted:=true;
      repeat
        temp:=sptr^.next^.num;sptr:=sptr^.next;
        start:=dl_delete(start,num);
        num:=temp;
      until sptr^.num = fptr^.num;
    end;

  procedure readvalues(var t:str255;var done,chn:boolean);

  {* Function     : Read the of the eneries for the data file and send out
                     a character string with all the four fields in it.  The
                     format of this string is compatible with that of the
                     Analyser module of the IONICS system
     Globals used  : freq,coststr
     Globals mod.  : None
     Routines used : None                                                  *}

    var a,b:real;
        s1,s2:str255;
        result,x,y:integer;
    begin
      x:=wherex;y:=wherey;
      repeat
        clreol;
        {$I-} readln(a,b); {$I+}          {read the values and check for}
        result:=ioresult;                 {errors                       }
        if result <> 0 then
          begin
            write(^g);write('Error');
            gotoxy(x,y);
          end;
      until result = 0;

      if (a = 0 ) and (b = 0 ) then       {end is indicated by both values}
        begin                             {being zero                      }
          done:=true;
          exit
```

```
    end;
  if a=0 then                          {if the first value is zero then}
    begin                              {a change in the constant value }
      chn:=true;exit;                  {is interpreted                 }
    end
  else chn:=false;
  str(a:10:3,s1);str(b:10:3,s2);
  if FREQ then
    t:=s1 + '  ' + conststr + '  ' + s2 + '  ' + '1.0'
  else
    t:=conststr + '  ' + '1.0' + '  ' + s1 + '  ' + s2;
end;

procedure auto_save(var next10:linepointer);

{ Function       : To do auto saving of the text entered after every 10
                   lines
  Globals used   : text (file variable)
  Globals mod.   : None
  Routines used  : None                                                 }

begin
  writeln('Doing auto save');
  while next10^.next <> nil do
    begin
      writeln(text,next10^.txt);
      next10:=next10^.next;
    end;
  flush(text);
end;

procedure enter;

{ Function       : To receive the entries and develop the link list
  Globals used   : start,prvsav,text
  Globals mod.   : None
  Routines used  : auto_save,readvalues                                 }

var ref,info:linepointer;
    num,vlu:real;
    finis,chn:boolean;
    rslt,x,y,lines:integer;
    txt:str255;

procedure value;

{ Function     : To read the constant value and store it as a character
                 string in conststr. This is local to the procedure enter
  Globals used : None
  Globals mod. : conststr
  Routines used: None                                                   }

begin
  write('Value of the ',param,': ');x:=wherex;y:=wherey;
  repeat
    clreol;vlu:=0;
    {$i-} readln(vlu); {$I+}
    rslt:=ioresult;
    if rslt <> 0 then
      begin
        write(^g);gotoxy(x,y);
      end;
  until (rslt = 0);
  if vlu = 0 then begin finis:=true;exit;end;
  str(vlu:8:3,conststr);
  writeln;write('Give the values of ');
  if freq then write('Z(kΩ) and T(°K) ')
  else write('Z(kΩ) and θ(°) ');
  writeln('in this order');writeln;
end;

begin
  num:=start^.num;clrscr;
  value;info:=nil;prvsav:=start;lines:=0;
  new(info);
  start^.next:=info;ref:=info;info^.prior:=start;
  rewrite(text);
  repeat
    num:=num+100;
    info^.num:=num;
    write(info^.num:6:0,': ');
    readvalues(txt,finis,chn);
    if chn then                        {if a change in the value of the constant}
      begin                            {is desired then read the new value and  }
        value;                         {restore its line number                 }
        num:=num-100;
      end
    else
      begin
        info^.txt:=txt;lines:=lines + 1;
        if lines = 10 then
          begin
            lines:=0;auto_save(prvsav);prvsav:=info;
          end;
        new(info);
        info^.next:=nil;
        info^.prior:=ref;
        ref^.next:=info;
        ref:=info;
      end;
  until (finis = true);
  info^.prior^.next:=nil;
  last:=info^.prior;
  auto_save(prvsav);
  close(text);
end;
```

```
function exist(filname:str255):boolean;

{  Function  :  To check if the file exist on disk
                if exist then true else false        }


      { if filname[i] in [')','(','?',';',',','\',':','"','''','/',
                          '.',']'
        then begin exist:= true;exit;end;
       assign(text,filname);{$i-}
       reset(text);{$I+}
       exist:=(ioresult=0);close(text);
     end;

procedure menu_0;

  {   Function:     Give the opening display          }

begin
  clrscr;drawObox(19,6,59,15);
  gotoxy(34,7); write('I O N I C S');
  gotoxy(27,9);write(',   Generate Module');
  gotoxy(21,12);write('Indian Institute of Technology Kanpur');
  gotoxy(38,14);write('1987');
  delay(2500);
end;                    {MENU_0}

procedure menu_1(var filename:str255);

  {   Function:     Menu for filename
                    loads the file name in fn (in upcase)
      Globals mod. : None                                  }

var i:integer;

begin
  clrscr;
  write('File name: ');                                 { get file name       }
  readln(filename);                                     { and validate it     }
  if length(filename)=0 then exit;
  if exist(filename) then
    begin
      newfile:=false;
      exit;
    end
  else
    newfile:=true;
  for i:= 1 to length(filename) do
     filename[i] := upcase(filename[i]);                { convert file name to }
                                                        { upper case letters   }
end;                                    {MENU_1}
```



```
procedure Fileheader;
  var info:linepointer;
  begin
    writeln;
    write('File header : ');                            { get file header     }
    read(hd);
    start:=nil;
    new(info);start:=info;
    info^.prior:=nil;info^.num:=100;
    info^.txt:=hd;
  end;

procedure menu_2( var choice:integer);

  {   Function:   Menu for choosing options for data entry          }

var ok:boolean;
begin
 clrscr;drawObox(4,2,29,4);
 gotoxy(6,3);write('IONICS Generate Module');
 gotoxy(8,6);write('Data File : ');
 write(fn);
 gotoxy(8,8);write('OPTIONS');writeln;writeln;
 writeln('          0. Quit');
 writeln;
 writeln('          1. Constant Frequency data');
 writeln('          2. Constant Temperature data');
 gotoxy(40,20);
 write('Your choice [0..2] : ');
 repeat
  gotoxy(61,20);clreol;
  {$I-} read(choice);{$I+}
  if (ioresult<>0) or (choice > 2) then
     write(^g);
 until (ioresult = 0) and (choice <= 2);
 freq:=(choice = 1);
 if freq then param:='Frequency(kHz)' else param := 'Temperature(°K)'
end;                                    {MENU_2}

 procedure menu_3(var choice:integer);

 { Function  : Menu for choosing the option of the Generate module   }

   var temp:string[80];
   begin
     clrscr;drawObox(4,2,29,4);
     gotoxy(6,3);writeln('IONICS Generate Module');writeln;
     getdir(0,temp);
     gotoxy(1,6);clreol;gotoxy(1,7);clreol;
     gotoxy(6,6);write('Current directory: ',temp);
```

```
    gotoxy(0,6);write('OPTIONS');writeln;writeln;
    writeln('          0. Quit');
    writeln;
    writeln('          1. Edit');
    writeln('          2. Enter');
    writeln('          3. Rename');
    writeln('          4. Delete');
    writeln;
    writeln('          5. Directory/New directory');
    gotoxy(40,20);
    write('Your choice [0..5] : ');
    choice:=0;
    repeat
      gotoxy(61,20);clreol;
      {$I-} read(choice);{$I+}
      if (ioresult<>0) or (choice > 5) then
        write(^g);
    until (ioresult = 0) and (choice <= 5);
  end;

procedure Display(var start:linepointer);

{ Function       : To display the contents of the loaded file
  Globals used   :start
  Globals mod.   :None
  Routines used  : find                                          }

  var temp:linepointer;
      c,cnt,num:real;
      x,y:integer;
  begin
    cnt:=20;num:=-10;                       {set the default values to display}
    write(linmsg);x:=wherex;y:=wherey;      {20 lines at a time, unless      }
    repeat                                  {otherwise specified             }
      gotoxy(x,y);clreol;
      {$I-} read(num,cnt); {$I+}            {if no num is specified then     }
    until ioresult = 0;                     { start at the beginning         }
    writeln;                                {if cnt not specified then       }
    c:=0;                                   {  display 20 lines at a time}
    if num < 0 then
      if num = -1 then temp:=start
      else  temp:=prv
    else
     begin
      if cnt = 0 then cnt:=last^.prior^.num;
      if num = 0 then temp:=last^.prior
        else temp:=find(num);
     end;
    while (temp <> nil) and (c <>cnt) do
      begin
        write(temp^.num:6:0,':   ');
        writeln(temp^.txt);c:=c+1;
        temp:=temp^.next;
      end;
    writeln;
```



```
  if temp = last then prv:=start else prv:=temp;
 end;

procedure Save(var f:filetype;var start.linepointer);

{ Function      : To save the edited file
  Globals used  : start
  Globals mod.  : None
  Routines used : None                                           }

  var ref:linepointer;
  begin
    ref:=start;
    writeln('Saving file ',fn);
    rewrite(f);
    while start <> nil do
      begin
        writeln(f,start^.txt);
        start:=start^.next;
      end;
    close(f);start:=ref;
  end;

function Load(var f:filetype;var pass:boolean):linepointer;

{ Function       : To load the file indicated in file variable f and pass
                   the pointer to the beginning of the file
  Globals used   : start
  Globals mod.   : start
  Routines used  : None                                                  }

  var ref,temp:linepointer;
      i:real;
  begin
    writeln('Loading file...');
    {$I-} reset(f); {$I+}                          {open the file for reading}
    if ioresult <> 0 then
      begin
        writeln('File not found');
        pass:=false;close(f);
        exit;
      end
    else
      pass:=true;
    while start <> nil do                          {get as much memory as possible}
      begin
        temp:=start^.next;
        dispose(start);
        start:=temp;
      end;
    last:=nil;new(temp);start:=temp;i:=100;temp^.prior:=nil;
    while not eof(f) do
      begin
        ref:=temp;
        readln(f,temp^.txt);temp^.txt:=temp^.txt + ^0;
```

```
        temp^.num:=i;i:=i+100;
        new(temp);
        if maxavail = 0 then      {keep checking for the available memory}
          begin
            writeln('Not enough memory.  Couldn''t load the entire file');
            close(f);load:=start;last:=ref;
            dispose(temp);
            exit;
          end;
        temp^.prior:=ref;
        ref^.next:=temp;
       end;
      ref^.next:=nil;last:=temp;
    load:=start;close(f);
  end;

procedure insert_in_line(var info:linepointer;var pos:integer);

{ Function        : To insert character in a line at position pos
                    Returns the position of the cursor in the line
   Globals used   : None
   Globals mod.   : None
   Routines used  : None                                              }

  var i:integer;
      instring:str255;
      ch:char;
  begin
    i:=i;
    repeat
      read(kbd,ch);
      if ch <> #27 then       {keep reading input till ESC is pressed}
        begin
          write(ch);
          if ch <> #8 then
            begin
              instring[i]:=ch;i:=i+1;
            end
          else i:=i-1;
        end;
    until (ch = #27);
    instring[0]:=chr(i-1);
    insert(instring,info^.txt,pos);pos:=pos+i-1;
  end;

procedure delete_from_line(var info:linepointer;pos:integer);

{  Function        : Delete characters from a line starting at position
                     pos till ESC is sent
   Globals used    : None
   Globals mod.    : None
   Routines used   : None                                             }

  var i:integer;
      ins:char;
```



```
  begin
    i:=0;
    repeat
     read(kbd,ins);
     if ins <> #27 then
       begin
         if ins in ['d','D'] then
           begin
             write('/');
             i:=i+1;
           end
       end;
    until ins = #27;
    delete(info^.txt,pos,i);
  end;

procedure Edit_line;

{  Function      : To monitor the commands in the edit mode for line
                   editing
                   'I' or 'i' means desire to insert in the line at the
                   position in the line indicated by the cursor
                   'D' or 'd' means desire to delete from the line starting
                   from the position indicated by the cursor
                   RETURN means simply display the chosen line
                   Any other key means display one character at a time
   Globals used  : prvline
   Globals mod.  : prvline
   Routines used : insert_in_line,delete_from_line                    }

  var temp:linepointer;
      i,j:integer;num:real;
      ch:char;
      x,y:integer;
  begin
    write(linemsg);x:=wherex;y:=wherey;
    repeat
      num:=-1;
      gotoxy(x,y);clreol;
      {$I-} read(num); {$I+}
    until ioresult = 0;
    writeln;
    if num < 0 then num:=prvline;prvline:=num;
    temp:=find(num);
    if temp = nil then exit;
    write(temp^.num:6:0,'  ');i:=1;
    repeat
      read(kbd,ch);
      if not(ch in ['i','I','D','d']) then
        begin
          if ch = #13 then            {display the entire line}
            begin
              for j:=i to length(temp^.txt) do
                write(temp^.txt[j]);
                i:=length(temp^.txt);
```

```
          end
        else
          write(temp^.txt[i]);i:=i+1;  {display one character}
      end
    else
      begin
        case ch of
          'i','I': begin
                     insert_in_line(temp,i);
                   end;
          'D','d': begin
                     delete_from_line(temp,i);
                   end
        end;
      end;
  until ((i=length(temp^.txt)+1) or (ch = #27));writeln;
end;

procedure REN;

{$ Function      : To get the file name to be renamed and rename it with the
                   new file name
   Globals used  : None
   Globals mod.  : None
   Routines used : exist                                                  $}

  var of1:text;
      old,new:str255;
  begin
    clrscr;
    repeat
      write('File name: ');
      readln(old);
      if length(old) = 0 then exit
      else
        if not(exist(old)) then
          writeln('File not found');
    until exist(old);
    repeat
      write('New file name: ');
      readln(new);
      if length(new) = 0 then exit
      else
        if exist(new) then
          writeln('File already exist in this name');
    until not(exist(new));
    assign(of1,old);
    rename(of1,new);
  end;
```



```
procedure DEL;

{$ Function      : To get the file name to be deleted and delete it
   Globals used  : None
   Globals mod.  : None
   Routines used : exist                                                  $}

  var temp:text;
      ch:char;
      i:integer;
      f:str255;
  begin
    clrscr;
    repeat
      write('File name: ');
      readln(f);
      if length(f) = 0 then exit
      else
        if not(exist(f)) then
          writeln('File not found');
    until exist(f);
    for i:=1 to length(f) do f[i]:=upcase(f[i]);
    assign(temp,f);
    write('Think again - do you wish to delete ',f,' (Y/N)');
    repeat
      read(kbd,ch);
    until ch in ['Y','y','N','n'];
    if ch in ['y','Y'] then erase(temp);
  end;

procedure directory;

{$ Function      : To do the management of the directory requests. This
                   gets the name of the file to be looked for and allow
                   wild card characters.  It also includes the routine
                   for change of directory.  If the directory is changed
                   then the new directory remains the active directory
                   until it is changed again.
   Globals used  : None
   Globals mod.  : None
   Routines used : None                                                   $}

type str12 = string[12];
var t:text;
    drive,dta_s,dta_f:integer;
    stglb,dir:string[80];
```

```
procedure DTA_addr;

{$ Function     : To get the current DTA address    $}

var i:integer;

  type
     dosreg = record
              case integer of
                1:( AX,BX,CX,DX,BP,SI,DI,DE,ES,FLAGS:integer);
                2: (AL,AH,BL,BH,CL,CH,DL,DH:byte);
              end;
  var
     reg:dosreg;
  begin
    with reg do
     begin
       AX:=$2F00;                    { call MS-DOS function $2F       }
       MSDOS(reg);                   { to get the current DTA address }
       dta_s:=ES;                    { address returned in ES:BX      }
       dta_f:=BX;
     end;
 end;

procedure set_DTA_addr;

{$ Function     : To set the DTA address to address stored in DTA_S:DTA_F  $}


  type
     dosreg = record
              case integer of
                1:( AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:integer);
                2: (AL,AH,BL,BH,CL,CH,DL,DH:byte);
              end;
  var
     reg:dosreg;
  begin
    with reg do
     begin
       AX:=$1A00;                    { call MS-DOS function $1A       }
       dta_addr;
       DS:=dta_s;                    {address to be set is in DS:DX   }
       DX:=dta_f;
       MSDOS(reg);
     end;
 end;

procedure search_first(var s:integer);

{$ Function       : To search for the first match
                    Takes in wild card characters     $}
```



```
  type
     dosreg = record
              case integer of
                1:( AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:integer);
                2: (AL,AH,BL,BH,CL,CH,DL,DH:byte);
              end;
  var
     reg:dosreg;
  begin
    with reg do
     begin
       AX:=$4E00;                    { call MS-DOS function $4E       }
       CX:=16;                       { file attribute to be used for  }
       DS:=seg(stglb);               { search (here NORMAL+Sub_DIR)   }
       DX:=ofs(stglb)+1;             { address of the ASCII7 string   }
                                     { with file name in DS:DX        }
       MSDOS(reg);
       s:=AX;                        { error code in AX =  2 if path invalid
     end;                            {                     18 if no match found
 end;

procedure search_next(var s:integer);

{$ Function       : To search for the next match
                    Any call to this must be preceded by a call to
                    search_first routine                              $}


  type
     dosreg = record
              case integer of
                1:( AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:integer);
                2: (AL,AH,BL,BH,CL,CH,DL,DH:byte);
              end;
  var
     reg:dosreg;
  begin
    with reg do
     begin
       AX:=$4F00;                    { call MS-DOS function $4F       }
       MSDOS(reg);
       s:=AX;                        { error code in AX =  18 if no match found
     end;
 end;

procedure Get_name(var na:str12);

{$ Function     : To get the name of the file whose FCB is in the current
                  DTA
                  The name and extension is found from offset 30 to 42 in the
                  FCB                                                        $}

  var i,l:integer;
      ch:char;
```

```
  begin
    i:=0;j:=1;
    repeat
      ch:=chr(mem[dta_s:(dta_ff29+j)]);
      nm[i]:=ch;
      i:=i+1;j:=j+1;
    until ch = #00;
    nm[0]:=chr(i-1);
  end;

procedure change_dir;

{* Function          : To change the active directory                    *}

  var st:string[80];
  begin
    getdir(drive,dir);
    writeln('Current directory: ',dir);
    write('New directory: ');
    readln(st);st:=st+#0;
    {$I-} chdir(st); {$I+}
    if ioresult <> 0 then write(^g);
    writeln;write('-');
  end;

procedure Setmask;

var path,dot:boolean;
    i:integer;
    l:char;

{* Function        : To set up the mask for the search along with full path
                     specifications.  It makes the ASCII input from the user
                     to an ASCIIZ string.

                     It takes in directory mask as any of the following:
                     'a:,b:,c:, etc. or a:\,b:\' etc.
                     '*.*' for the current directory
                     '\' for the root directory of the default drive
                     '.....\.*' for the sub-directories
                     any wild card character without or with a valid path.

   Globals used    : stglb,dir
   Globals mod.    : stglb
   Routines used   : None                                                 *}

  begin
    getdir(drive,dir);
    write('Mask: ');read(stglb);
    i:=0;dot:=false;path:=false;
    while (i < length(stglb)) and  (path = false) do
      begin                        {find if the path was give by locating '\'}
        i:= i + 1;                 {if yes then find if dot is in the string }
        if stglb[i] = '\' then
          begin
            path:=true;
            i:=length(stglb);
          end
        else
          i:=i+1;
      end;
    if pos('.',stglb) <> 0 then dot:=true
    else dot:=false;
    l:=stglb[length(stglb)];
    if not((path and dot) or (not(path) and dot)) then
      begin
        if path and not(dot) then
          if not(l in ['*','\']) then stglb:=stglb + '\*.*'
          else if l = '\' then stglb:=stglb + '*.*';
        if not(path) and not(dot) then
          if l = ':'then stglb:=stglb + '\*.*'
          else stglb:=dir+stglb+'\*.*';
      end
    else
      if length(stglb) in [1,2] then
        if (stglb = '.') or (stglb = '..') then
          if l = '\' then stglb:=stglb + '*.*'
          else stglb:=stglb + '\*.*';
     stglb:=stglb + #0;
     writeln;
  end;

procedure Call_dir;

{* Function        : To call the specified directory with the give mask
                     and display the files found (five files per line)
   Globals used    : stglb
   Globals mod.    : None
   Routines used   : setmask,search_first,search_next,get_name            *}

  var s,count,offset:integer;
      fn:str12;

  begin
    setmask;count:=0;
    writeln;
    writeln(' ','Files of: ',stglb);
    search_first(s);
    get_name(fn);
    if count mod 5 = 0 then writeln;
    if s <> 18 then
      begin
        write('   '+fn);count:=count+1;
      end;
    for offset:=1 to 12-length(fn) do write(' ');
    if s = 18 then                        {stop if the first match is not found}
      begin
        writeln;writeln('   No file(s) found');
      end
    else
```

```
      while s <> 18 do
        begin
          search_next(s);     {get the next match and display the names till}
          get_name(fn);       {no match is reported                        }
          if count mod 5 = 0 then writeln;
          if s <> 18 then begin write('   '+fn);count:=count+1; end;
          for offset:=1 to 12-length(fn) do write(' ');
        end;
      writeln;write('-');
  end;

procedure comnd;

{* Function         : To process the commands for directory handling  *}

  var ch:char;
      done:boolean;

  procedure Help;
    begin
      write('D  for Directory      N  for New directory      ');
                                          writeln('E  for End of the section');

      writeln;write('-');
    end;

  begin
    done:=false;
    writeln;write('-');
    repeat
    read(kbd,ch);
    case ch of
      'd','D':call_dir;
      'n','N':begin change_dir;done:=true;end;
      'e','E':done:=true
       else help;
    end;
    until done;
  end;

begin
  drive:=0;
  set_dta_addr;
  writeln;
  comnd;
end;

procedure screenout;

{* Function      : To make the screen for the edit mode of GENERATE   *}

  begin
    gotoxy(1,24);write('S');gotoxy(15,24);write('V');
    gotoxy(29,24);write('E');
    gotoxy(44,24);write('I');gotoxy(59,24);write('D');
    gotoxy(74,24);write('Q');
```



```
    textbackground(white);textcolor(black);
    gotoxy(3,24);write('Save');gotoxy(17,24);write('View');
    gotoxy(31,24);write('Edit');
    gotoxy(46,24);write('Insert');gotoxy(61,24);write('Delete');
    gotoxy(76,24);write('Quit');textcolor(white);textbackground(black);
    gotoxy(1,1);
  end;

procedure editmode;

{* Function       : Master controller of the edit mode
   Globals used   : start
   Globals mod.   : None
   Routines used  : remove,display,save,insertline,edit_line       *}

 var ch:char;
     finis,saved,d:boolean;

 procedure savmsg;
   var ch:char;
   begin
     writeln;
     write('Changes have been made. Do you wish to save the edited file(Y/N)');
     repeat
       read(kbd,ch);
     until ch in ['n','N','Y','y'];
     writeln(' ',upcase(ch));
     if ch in ['y','Y'] then save(text,start);
   end;

 begin
   screenout;window(1,1,80,23);
   writeln;writeln;writeln;write('*');
   prv:=start;prvline:=0;finis:=false;
   saved:=true;

   repeat
     read(kbd,ch);
     case ch of
        'd','D': begin remove(d);saved:=not(d);write('*');end;
        'v','V': begin display(start);write('*');end;
        's','S': begin saved:=true;save(text,start);write('*');end;
        'q','Q': begin if not(saved) then savmsg;finis:=true;end;
        'i','I': begin saved:=false;insertline;writeln;write('*');end;
        'e','E': begin saved:=false;edit_line;write('*');end
      end;
   until finis;
   window(1,1,80,24);clrscr;
 end;

function Editmsg:integer;

{* Function       : To send a message if wrong action is desired on file
                    and take appropriate action
   Globals used   : None
```

```
  Routines used  : None                                          *)

  var ch:char;
  begin
    writeln;
    writeln('File ',fn,' does not exist');
    write('Do you wish to create a new file(Y/N): ');
    repeat
      read(kbd,ch);
    until ch in ['y','Y','n','N'];
    if ch in ['y','Y'] then
      begin
        editmsg:=2;
        clrscr;
      end
    else editmsg:=-1;
  end;

function Entermsg:integer;

(*  Function        : To send a message if wrong action is desired on a file
    Globals used    : None
    Globals mod.    : None
    Routines used   : None                                       *)

  var ch:char;
  begin
    writeln;
    writeln('File ',fn,' already exist');
    write('Did you intend to edit the file(Y/N): ');
    repeat
      read(kbd,ch);
    until ch in ['y','Y','n','N'];
    if ch in ['y','Y'] then
      begin
        entermsg:=1;
        clrscr
      end
    else entermsg:=-1;
  end;

begin
  check:=cnt;
  getdir(0,dir_ref);
  start:=nil;
  last:=nil;
  done:=false;clrscr;
  clrscr;menu_0;
  repeat
    done:=false;
    menu_3(op);
    if op in [1,2] then
      begin
        menu_1(fn);
```



```
        begin
          if (newfile and (op = 1) then
            op:=editmsg
          else
            begin
              if (not(newfile)) and (op = 2) then
                  op:=entermsg;
            end;
        end
      else op:= -1;
    end;

  case op of
    0  :begin done:=true;clrscr;end;
    2  :begin
          assign(text,fn);
          fileheader;
          menu_2(op);
          if op <> 0 then  enter;
        end;
    1  :begin
          clrscr;
          writeln('Edit mode');
          start:=load(text,sts);
          if sts = true then editmode else delay (2500);
        end;
    3  :ren;
    4  :del;
    5  :directory
  end;
until done;
chdir(dir_ref);
cnt:=check;
if check > 127 then
  begin
    assign(text,dir_ref+'\'+'ionics.com');
    execute(text);
  end;
end.
```

```
{!!!
```

```
                    begin
                      writeln(msg1);
                      ok:=false;
                      goto 100
                    end;
                  if abs(values[i]-t)<taccglb then
                    begin
                      nl:=nl+1;dataglb[1,1]:=z;dataglb[1,2]:=th;l:=l+1;
                      if l>200 then
                      begin
                        writeln(msg2);
                        ok:=false;
                        goto 100
                      end
                    end
                end;
              if i=1 then
                begin
                  dsa[i]:=1;
                  dea[i]:=nl;
                  c:=c+nl;
                end
              else
                begin
                  dsa[i]:=c+1;dea[i]:=nl+c;c:=c+nl;
                end;
              if (nl=0) or (nl<3) then
                begin
                  ok:=false;
                  writeln(msg3,values[i]:7:2);
                end;
            reset(datfile);readln(datfile);
            end;
        end;

    2,5:begin
          if (op=2) or (op=4) then
            begin
             for i:=1 to nofcatset do
             begin
              nl:=0;
              while not eof(datfile) do
               begin
                 {$I-} readln(datfile,t,f,z,th);{$I+}
                 if ioresult<>0 then
                   begin
                     writeln(msg1);
                     ok:=false;
                     goto 100
                   end;
                 if abs(values[i]-f)<(faccglb*1E-03) then
                  begin
                   nl:=nl+1;dataglb[1,2]:=z;dataglb[1,1]:=abs(t);l:=l+1;
                   if l>200 then
                      begin
```



```
                        writeln(msg2);
                        ok:=false;
                        goto 100
                      end
                  end
                end;
              if i=1 then
                begin
                  dsa[i]:=1;
                  dea[i]:=nl;
                  c:=c+nl;
                end
              else
                begin
                  dsa[i]:=c+1;dea[i]:=nl+c;c:=c+nl;
                end;
              if (nl=0) or (nl<3) then
                begin
                  ok:=false;
                  writeln(msg3,values[i]:7:2);
                end;
            reset(datfile);readln(datfile);
            end
        end;
      end;
    9:begin
        while not eof(datfile) do
          begin
            {$I-} readln(datfile,t,f);{$I+}
            if ioresult<>0 then
              begin
                writeln(msg1);
                ok:=false;
                goto 100
              end;
              dataglb[1,1]:=t;dataglb[1,2]:=f;l:=l+1;
              if l>200 then
                begin
                  writeln(msg2);
                  ok:=false;
                  goto 100;
                end
            end;
          dea[1]:=1;
        end
    end;  { of case }
    100:close(datfile);
end;

overlay procedure datainterpret(num,code:integer);

{  CONVERT DATA FOR PLOTTING BY CALCULATING
                              Z Cos[0] and Z Sin [0]
                              log(r) and 1000/T              AS REQUIRED
```

```
    Globals modified:       num,dataglb,dea,dsa
    Routines used:          None                                  *)

  var i:integer;
      t:real;

  begin
    if code=1 then
      begin
        for i:=1 to dea[num] do
          begin
            t:=dataglb[i,1];
            dataglb[i,1]:=t*abs(cos(dataglb[i,2]*dtor));
            dataglb[i,2]:=t*abs(sin(dataglb[i,2]*dtor));
          end
      end
    else
      begin
        for i:=1 to dea[num] do
          begin
            dataglb[i,2]:=log(la_ratio/dataglb[i,2]);
            dataglb[i,1]:=1000.0/(dataglb[i,1]);
          end
      end
  end;

overlay procedure DOCIRFIT(ip,fp,crv:integer;var r,xc,yc:real;var err:boolean);

(*    Function :    To find the centre and R of curve no.crv between
                    the initial and final points
      Globals used: dataglb,dsa
          modified: None
      Routines used: None                                         *)

  label 100,10,2000,2100;
  var
      rt,dt:array[1..25] of real;
      counter,c,i,j:integer;
      acc,t,igx,igy,min,a,b:real;
      gxy:array[1..25,1..2] of real;
      one,er:boolean;
      rpl,rply:char;

procedure GETCENT;

(*  A routine to used by DOCIRFIT to internaly get the circle parameters *)

    var mp,mp2,i:integer;
        ipp,fpp,i1,i2,i3:integer;
        xc1,xc2,yc1,yc2,m1,m2:real;
        dy1,dy2,x1,y1,x2,y2,x3,y3:real;
        xcent,ycent:array[1..2] of real;
    begin
      ipp:=ip;fpp:=fp;
      mp:=round((fpp-ipp+1)/2);   { get the first,last and the midpoints }
      mp2:=round(mp/2);           { of the two quaters of the semi circle}
      err:=false;
      for i:=1 to 2 do
        begin
          i1:=dsa[crv]+ipp-1;     { draw the perpendicular bisectors and }
          i2:=dsa[crv]+mp2-1;     { get the initial guess of the center  }
          i3:=dsa[crv]+mp-1;
          x1:=dataglb[i1,1];  y1:=dataglb[i1,2];
          x2:=dataglb[i2,1];  y2:=dataglb[i2,2];
          x3:=dataglb[i3,1];  y3:=dataglb[i3,2];
          dy1:=y1-y2;dy2:=y2-y3;
          gotoxy(1,24);clreol;
          gotoxy(1,23);clreol;
          if (dy1=0) or (dy2=0) then
            begin
              write('Data not well behaved!        Hit a key');
              repeat until keypressed;
              err:=true;
              exit
            end;
          m1:=-(x1-x2)/dy1;  m2:=-(x2-x3)/dy2;
          xc1:=(x1+x2)/2; yc1:=(y1+y2)/2;
          xc2:=(x2+x3)/2; yc2:=(y2+y3)/2;
          xcent[i]:=((yc1-m1*xc1)-(yc2-m2*xc2))/(m2-m1);   { first guess }
          ycent[i]:=m1*xcent[i]+yc1-m1*xc1;
          ipp:=mp; mp2:=round((fpp-mp+1)/2)+mp; mp:=fpp;
        end;
      igx:=(xcent[1]+xcent[2])/2;
      igy:=(ycent[1]+ycent[2])/2;
    end;
  begin
    err:=false;   { first approach }
    2100:igx:=(dataglb[ip+dsa[crv]-1,1]+dataglb[fp+dsa[crv]-1,1])/2;
    igy:=(dataglb[ip+dsa[crv]-1,2]+dataglb[fp+dsa[crv]-1,2])/2;
    one:=false;gotoxy(78,1);write('M1');

    2000:acc:=0.08;counter:=0;gotoxy(1,1);
    gotoxy(1,24);clreol;gotoxy(1,23);clreol;
    write('    R= ');gotoxy(25,23);
    write('Cent.: ');gotoxy(60,24);clreol;
    write('Please wait');
    gotoxy(1,24);write('Error= ');
    10:a:=acc*2;b:=acc*2;i:=1;          { develope the grid around the  }
    repeat                              { current centre                }
      for j:=1 to 5 do
        begin
          gxy[i,1]:=igx+igx*A;
          gxy[i,2]:=igy+igx*B;
          B:=B-acc;
          i:=i+1;
        end;
      B:=acc*2;A:=A-acc;
    until (i)=25);                      { get the R and deviations for }
    for i:=1 to 25 do                   { each of the  grid points     }
     begin
```

```
    dt[i]:=0;t:=0;
    for j:=(dsa[crv]+ip-1) to (dsa[crv]+fp-1) do
      t:=t+sqrt(sqr(gxy[i,1]-datag]b[j,1])+sqr(gxy[i,2]-datag]b[j,2]));
    rt[i]:=t/(fp-ip+1);
    for j:=(dsa[crv]+ip-1) to (dsa[crv]+fp-1) do
      dt[i]:=dt[i]+sqr(rt[i]-sqrt(sqr(gxy[i,1]-datag]b[j,1])
            +sqr(gxy[i,2]-datag]b[j,2])));
  end;
  min:=dt[1];c:=1;                    { find the least deviation }
  for i:=2 to 25 do
    begin
      if dt[i]<min then
        begin
          c:=i;
          min:=dt[i];
        end;
    end;
  if counter = 15 then                { check for 15 iterations }
    begin
      gotoxy(1,24);clreol;gotoxy(1,23);clreol;
      write('No convergence in 15 iterations!');
      gotoxy(1,24);
      if one then
        begin
          gotoxy(36,23);write('Hit any key');
          repeat until keypressed;
          err:=true;
          exit
        end;
      write('Want to try another approach(Y/N) : ');
      read(kbd,rply);
      if (rply in ['y','Y']) then
        begin
          one:=true;gotoxy(78,1);write('M2');
          getcent;                    { for approach 2 }
          if err then exit;
          goto 2000;
        end
      else
        begin
          err:=true;exit;
        end;
    end;

  if (igx<>0) and (igy<>0) then       { check for convergence }
   begin
    if (c in [1,2,3,4,5,6,10,11,15,16,20,21,22,23,24,25]) then
     begin
      counter:=counter+1;
      if dt[c]=0.0 then
          goto 100;
      igx:=gxy[c,1];
      igy:=gxy[c,2];
      gotoxy(8,23);write(rt[c]:11);
      if keypressed then
```



```
          begin
            read(kbd,rpl);
            if rpl in ['s','S'] then
              begin
                if not(one) then
                  begin
                    one:=true;
                    gotoxy(78,1);write('M2');
                    getcent;                { for approach 2 }
                    if err then exit;
                      goto 2000;
                  end
                else goto 2100
              end
          end;
        gotoxy(32,23);write('(',gxy[c,1]:11,',',gxy[c,2]:11,')');
        gotoxy(9,24);write(acc*100:6:2,' %');
        goto 10;
      end;
    end;
    gotoxy(8,23);write(rt[c]:11);
    gotoxy(32,23);write('(',gxy[c,1]:11,',',gxy[c,2]:11,')');
    gotoxy(9,24);write(acc*100:6:2,' %');
    if acc<0.03 then
      begin
        gotoxy(60,24);
        write('Approve(Y/N)?');
        read kbd,rpl);
        case rpl of
          'y','Y': begin
                     counter:=0;
                     gotoxy(60,24);
                     write('Please wait   ');
                     acc:=acc/2;
                     goto 10;
                   end;
          's','S': begin
                     if not(one) then
                       begin
                         one:=true;
                         gotoxy(78,1);write('M2');
                         getcent;                { for approach 2 }
                         if err then exit;
                         goto 2000;
                       end
                     else goto 2100
                   end
        end;{of case}
      end;
    gotoxy(60,24);
    write('Please wait      ');
    100:xc:=gxy[c,1];yc:=gxy[c,2]; r:=rt[c];
  end;

overlay procedure arrange(var nums:plotarray;num:integer);
```

```
{*  Function:       Sorts datglb in ascendin order of x co-ord.
    Globals used:   dataglb,num,dea,dsa
        modified:   dataglb
    Routines used:  None                        *}

  var temp1,temp2:real;
      k,m,i,j,pass,gap:integer;
  procedure flipflop;
    begin
      temp1:=nums[j,1];temp2:=nums[j,2];
      nums[j,1]:=nums[j+gap,1];
      nums[j,2]:=nums[j+gap,2];
      nums[j+gap,1]:=temp1;nums[j+gap,2]:=temp2;
    end;
  begin
    for k:=1 to num do
    begin
      m:=dea[k]-dsa[k]+1;
      gap:=m div 2;
      while gap>0 do
        begin
          for i:=(gap+dsa[k]) to dea[k] do
            begin
              j:=i-gap;
              while j>(dsa[k]-1) do
                begin
                  if nums[j,1]>nums[j+gap,1] then
                    begin
                      flipflop;
                      j:=j-gap;
                    end
                  else j:=dsa[k]-1;
                end;
            end;
          gap:=gap div 2;
        end;
    end;
  end;

procedure readopen(filename:name;var ok:boolean);

{*  Function:           Opens file for reading
    Globals used:       None
          modified:     ok
    Routines used:      None                    *}

  begin
      if filename[1] in ['<','>','?','!',',','\',':','"','''','/',
                         '|',';',' ','*','+','=','{','}']
        then begin ok:=false;exit;end;
      assign(datfile,filename);
      {$i-} reset(datfile)
       {$i+}; ok:=(ioresult=0)
  end;
```



```
procedure doplot(w:integer;num,op:integer;wait:boolean;M:integer);

{*  Function:           Plots the data in datglb
    Globals used:       dataglb,hd,num,op,dea,dsa
          modified:     None
    Routines used:      None                      *}
  const a=1.4545;
  var points:plotarray;
      i,j,n,l:integer;
      ratio,t:real;
  begin
   findworld(w,dataglb,M,1,1);          {Define the best World}
   if op=1 then
     begin
       with world[w] do
         begin
           if x1<>0 then                {If Xmin<>0 and Xmin&Xmax not(0 }
             if (x2/x1)>0) then         {then start x axis from zero    }
                 x1:=0;
           if y1<>0 then
             if (y2/y1)>0) then
                 y2:=0;
           ratio:=(x2-x1)/(y1-y2);
           if ratio>a then
             y1:=y2+(x2-x1)/a           {Scling of the axis}
           else
             if ratio<a then
               x2:=(y1-y2)*a+x1;
         end;
     end;
   with world[w] do
     begin
       t:=y1;
       y1:=y2;
       y2:=t
     end;
   selectworld(w);
   defineheader(w,hd);setheaderon;
   selectwindow(w);
   drawborder;drawaxis(0,0,0,0,0,0,0,0,0,false);
   for i:=1 to num do                   {Load the data set in the array}
     begin
       M:=dea[i]-dsa[i]+1;l:=1;
       for j:=dsa[i] to dea[i] do
       begin
         points[l,1]:=dataglb[j,1];
         points[l,2]:=dataglb[j,2];
         l:=l+1;
       end;
       if n>=2 then
         begin
           drawaxis(0,0,0,0,0,0,0,0,0,false);
           drawpolygon(points,1,-n,-1,2,0);
         end;
```

```
          setlinestyle(0);
        end;
      if nofcir > 0 then
      for i:=1 to nofcir do
          drwcir(rglb[i],xcglb[i],ycglb[i],false);
      if (r='y') or (r='Y') then
        begin
{$          screendump;                    {Call interupt 5 for screen dump}
 $}        drwRAM;
          hardcopy(false,6);
          clearscreen;
          selectscreen(1);
       end;
      repeat until keypressed;
    end;

procedure screenio(num:integer);

  {$  Function:     Display the Data set symbols on the screen
      Gobals used:  num
         modified:  None
      Routines used:None                                        $}

    var p:integer;
        directmodeloc:boolean;
    procedure clear(i1,i2:integer);
       var j1:integer;
       begin
         for j1:=i1 to 80 do write(' ');
         gotoxy(i1,i2);
       end;
   begin
     gotoxy(71,7);clear(71,7);
     directmodeloc:=directmodeglb;
     defineworld(2,0,0,80,25);
     selectworld(2);selectwindow(2);
     write('Data Set');
     gotoxy(69,9);clear(69,9);
     directmodeglb:=true;
     for p:=1 to num do
       begin
         case p of
           1:begin drawcross(round(7.12*xmaxglb),round(0.34*ymaxglb),2);
                   gotoxy(74,9);write(': ');gotoxy(77,9);
                   write(P);
             end;
           2:begin drawcrossdiag(round(7.12*xmaxglb),round(0.38*ymaxglb),2);
                   gotoxy(74,10);write(': ');
                   gotoxy(77,10);
                   write(p);
             end;
           3:begin drawsquareC(round(7.096*xmaxglb),round(0.415*ymaxglb),
                        round(7.142*xmaxglb),round(0.4257*ymaxglb),false);
                   gotoxy(74,11);write(': ');gotoxy(77,11);write(p);
             end;
```



```
           4:begin drawsquareC(round(7.096*xmaxglb),round(0.472*ymaxglb),
                        round(7.142*xmaxglb),round(0.452*ymaxglb),true);
                   gotoxy(74,12);
                   write(': ');
                   gotoxy(77,12);write(p);
             end;
           5:begin drawdiamond(round(7.12*xmaxglb),round(0.5*ymaxglb),2);
                   gotoxy(74,13);write(': ');
                   gotoxy(77,13);
                   write(p);
             end;
           6:begin drawwye(round(7.12*xmaxglb),round(0.543*ymaxglb),2);
                   gotoxy(74,14);write(': ');
                   gotoxy(77,14);
                   write(p);
             end;
           7:begin drawstar(round(7.12*xmaxglb),round(0.583*ymaxglb),2);
                   gotoxy(74,15);write(': ');
                   gotoxy(77,15);
                   write(p);
             end;
           8:begin drawcircledirect(round(7.12*xmaxglb),
                                round(0.623*ymaxglb),2,false);gotoxy(74,16);
                   write(': ');
                   gotoxy(77,16);write(p);end
         end;
       end;
       directmodeglb:=directmodeloc;
     end;

procedure crvno(var crv:integer;num:integer);

  {$  Function:      Get the data set No. from the user
      Globals used:  num
         modified:   None
      Routine used:  None                                       $}

    begin
     gotoxy(1,24);clreol;
     gotoxy(1,23);clreol;
     write('Curve No.: ');
     repeat
       gotoxy(11,23);clreol;{$I-} readln(crv);{$I+}
     until ((ioresult = 0) and  (crv<=num));
   end;

procedure rub(num,crv:integer;clr:boolean);

  {$   Function:       To rub all the data points except the one choosen
       Globals used:   num
          modified:    None
       Routines used:  None                                                $}

    var bl:plotarray;
        l,p,i,nopt:integer;bll:plotarray;
```

```
begin
 if num>1 then
   begin
     setlinestyle(1);selectwindow(3);
     selectworld(1);
     drawaxis(0,0,0,0,0,0,0,0,0,false);
     setcolorblack;                    (Rub all data set except crv)
     if not(clr) then
       begin
        for p:=1 to num do
          begin
            l:=1;
            if p<>crv then
            begin
              for i:=dsa[p] to dea[p] do
                begin
                  b11[1,1]:=dataglb[i,1];
                  b11[1,2]:=dataglb[i,2];l:=l+1;
                end;
              nopt:=dea[p]-dsa[p]+1;
              resetaxis;drawpolygon(b11,1,-nopt,-p,2,0)
            end;
          end
       end
     else
       begin
         l:=1;
         for i:=dsa[prvcrv] to dea[prvcrv] do
           begin
             b11[1,1]:=dataglb[i,1];
             b11[1,2]:=dataglb[i,2];l:=l+1;
           end;
         nopt:=dea[prvcrv]-dsa[prvcrv]+1;
         resetaxis;drawpolygon(b11,1,-nopt,-prvcrv,2,0)
       end;
     resetaxis;setlinestyle(1);
     if noflines<>0 then
       drawpolygon(lines,1,-(noflines*3),0,0,0);  ( Rub the lines drawn)
     if nofcir<>0 then
       for l:=1 to nofcir do
        begin
         setlinestyle(0);
         drwcir(rglb[l],xcglb[l],ycglb[l],false);
         setlinestyle(1);
        end;
     l:=1;
     for i:= dsa[crv] to dea[crv] do
       begin
         b11[1,1]:=dataglb[i,1];b11[1,2]:=dataglb[i,2];
         l:=l+1;
       end;
     nopt:=dea[crv]-dsa[crv]+1;
     setcolorwhite;setlinestyle(0);
     drawaxis(0,0,0,0,0,0,0,0,0,false);
     drawpolygon(b11,1,-nopt,-crv,2,0);      (Redraw the choosen curve)
```



```
       prvcrv:=crv;
     end;
   end;

procedure setlimits(crv:integer;var ip,fp:integer);

  (*   Function:       Set up the initial and the final points for curve
                       fitting
       Globals used:   None
           modified:   None
       Rontines used:  None                                              *)

     label 20,40;
     const msg1 = 'X co_ord.= ';
           msg2 = 'Y co_ord.= ';
           corr = 'Correct?';
     var r:char;
         n:integer;
     begin
     n:=dea[crv]-dsa[crv]+1;
     20:gotoxy(1,23);clreol;
     write('Initial point: ');
     repeat
       gotoxy(15,23);clreol;{$I-} read(ip);{$I+}
       if ip<=0 then
         begin
           gotoxy(1,23);clreol;gotoxy(1,25);clreol;
           exit;
         end;
     until ((ioresult = 0)  and (ip<=n));
     gotoxy(40,23);
     write(msg1,dataglb[dsa[crv]+ip-1,1]:7:2);
     gotoxy(1,24);clreol;gotoxy(40,24);
     write(msg2,dataglb[dsa[crv]+ip-1,2]:7:2);
     gotoxy(1,25);write(corr);
     gotoxy(69,24);
     repeat
       read(kbd,r);
     until (r in ['y','n','N','Y']);
     if (r='n') or (r='N') then
       begin
         gotoxy(1,24);clreol;
         gotoxy(1,25);clreol;goto 20;
       end;
     gotoxy(1,25);clreol;gotoxy(1,24);clreol;
     40:gotoxy(1,23);clreol;
     write('Final point: ');
     gotoxy(14,23);{$I-} read(fp);{$I+}
     if fp<=0 then goto 20;
     if (fp<ip) or (fp=ip) or (fp>n) or (ioresult<>0) then
       goto 40;
     gotoxy(40,23);
     write(msg1,dataglb[dsa[crv]+fp-1,1]:7:2);
     gotoxy(1,24);clreol;gotoxy(40,24);
     write(msg2,dataglb[dsa[crv]+fp-1,2]:7:2);
```

```
      gotoxy(1,25);write(corr);
      gotoxy(69,24);
      repeat
        read(kbd,r);
      until (r in ['n','y','N','Y']);
      if (r='n') or (r='N') then
        begin
          gotoxy(1,24);clreol;gotoxy(1,25);clreol;
          goto 40;
        end;
      gotoxy(1,25);clreol;
    end;

procedure getequation(crv,ip,fp:integer;var M,C:real);

  {*    Function:      Set up straight line equation
        Globals used:  None
           modified:   None
        Routines used: None                          *}

    var j,npt:integer;
        sumx,sumy,sumxy,sumxsq,numr,deno:real;
    begin
      sumx:=0.0;sumy:=0.0;sumxsq:=0.0;sumxy:=0.0;selectworld(1);
      for j:=(dsa[crv]+ip-1) to (dsa[crv]+fp-1) do
        begin
          sumx:=dataglb[j,1]+sumx;
          sumy:=dataglb[j,2]+sumy;
          sumxsq:=dataglb[j,1]*dataglb[j,1]+sumxsq;
          sumxy:=dataglb[j,1]*dataglb[j,2]+sumxy;
        end;
      npt:=fp-ip+1;
      numr:=(npt*sumxy)-(sumx*sumy);
      deno:=npt*sumxsq-sumx*sumx;
      M:=(numr/deno);C:=(sumy-M*sumx)/npt;
    end;

procedure drwlin(crv,ip,fp:integer;M,C:real;var lines:plotarray;
                 var noflines:integer);

  {*    Function:      Draw the best fit line
        Global used:   None
           modified:   None
        Routines used: None                  *}

    var b2:plotarray;
    begin
      b2[1,1]:=dataglb[dsa[crv]+ip-1,1];b2[1,2]:=M*b2[1,1]+C;
      b2[2,1]:=dataglb[dsa[crv]+fp-1,1];b2[2,2]:=M*b2[2,1]+C;
      b2[3,1]:=b2[2,1];              b2[3,2]:=b2[2,2];
      lines[r1+1,1]:=dataglb[dsa[crv]+ip-1,1];lines[r1+1,2]:=M*lines[r1+1,1]+C;
      lines[r1+2,1]:=dataglb[dsa[crv]+fp-1,1];lines[r1+2,2]:=M*lines[r1+2,1]+C;
      lines[r1+3,1]:=lines[r1+2,1];        lines[r1+3,2]:=lines[r1+2,2];
      r1:=r1+3;p1:=p1+1;
      setlinestyle(0);
```



```
      selectwindow(3);
      selectworld(1);
      drawaxis(0,0,0,0,0,0,0,0,false);
      setlinestyle(1);drawpolygon(b2,1,-3,0,0,0);setlinestyle(0);
      gotoxy(1,24);clreol;
      gotoxy(1,23);clreol;
      gotoxy(24,23);
      write('Y=',M:7:2,'*X+(',c:7:2,')');
      repeat until keypressed;
      noflines:=p1-1;
    end;

procedure stlinefit(op:integer;num:integer;
                    templeng:integer);

  {*   Funtion:       Control the fitting of the best st.line
       Globals used:  fn.op,num,hd
           modified:  None
       Routines used: crvno,rub,screenio,setlimits,getequation,drwlin,
                      hardcopysan                                    *}

    label 10;
    var crv,ip,fp,g,noflines:integer;
        m,c:real;
        lines:plotarray;
        r:char;clr:boolean;
  begin
    r1:=0;prvcrv:=0; p1:=1;noflines:=1;nofclr:=0;clr:=false;
    for g:=1 to 24 do
      begin
        lines[g,1]:=0.0;lines[g,2]:=0.0;
      end;
    screenio(num);
    10:gotoxy(71,0);
    if not(rdc) then crvno(crv,num)
    else crv:=1;
    if (crv<=0) then
      begin
        hardcopysan(num,op,lines,noflines,templeng);exit
      end;
    if crv<>prvcrv then
     begin
      rub(num,crv,clr);
      clr:=true;
     end;
    setlimits(crv,ip,fp);
    if (ip<=0) or (fp<=0) then
      goto 10;
    getequation(crv,ip,fp,m,c);
    drwlin(crv,ip,fp,m,c,lines,noflines);
    gotoxy(1,24);clreol;
    gotoxy(1,23);clreol;
    write('More lines to be fitted?');
    repeat
      gotoxy(25,23);read(kbd,r);
```

```
      until (r in ['n','y','N','Y']);
      if (r='y') or (r='Y') then goto 10;
      hardcopysan(num,op,lines,noflines,templeng);
    end;

procedure circlefit(op,num:integer;templeng:integer);

(*   Function    :    To fit the circles
     Globals used:    nofcir,crv,xcglb,ycglb,rglb
        modified :    xcglb,ycglb,rglb
     Routines used :  rub,hardcopysan                          *)

  label 5,10,20,30,40,100;
  var ip,fp:integer;
      r,xc,yc:real;
      rp:char;
      noflines:integer;
      lines:plotarray;
      e,clr:boolean;

  begin
    nofcir:=0;noflines:=0;clr:=false;
    screenio(num);
    5:crvno(crv,num);
    if (crv<=0) then goto 100;
    rub(num,crv,clr);clr:=true;
    setlimits(crv,ip,fp);
    if (ip<=0) or (fp<=0) then goto 5;
    docirfit(ip,fp,crv,r,xc,yc,e);
    if e then goto 30;
    selectwindow(3);selectworld(1);
    drawaxis(0,0,0,0,0,0,0,0,false);
    drwcir(r,xc,yc,true);nofcir:=nofcir+1;
    rglb[nofcir]:=r;  xcglb[nofcir]:=xc;    ycglb[nofcir]:=yc;
    30:gotoxy(1,24);clreol;gotoxy(1,23);
    clreol;write('More circles to be fitted(Y/N)?');
    20:{$I-}read(kbd,rp);{$I+}
    if ioresult<>0 then goto 20;
    case rp of
      'Y','y': goto 5;
      'N','n': goto 10
    end;
    goto 20;
    10:hardcopysan(num,op,lines,noflines,templeng);
    100:
  end;

procedure plotcontroller(op:integer;num:integer);

  (*   Function:       Controlls the plotting sub-module
       Globals used:   op,num,hd,dea,dsa,lines noflines,
           modified:   None
       Routines used:  doplot,hardcpysan,stlinefit,circlefit     *)

    label 10,20,100;
```



```
    var   r:char;
          ml,ii,noflines:integer;lines:plotarray;
    begin
      noflines:=0;
      for ii:=1 to 3 do
        begin
          lines[ii,1]:=0.0;lines[ii,2]:=0.0;
        end;
      initgraphic;clearscreen;
      definewindow(1,0,0,Xmaxglb,Ymaxglb);
      definewindow(2,0,0,Xmaxglb,Ymaxglb);
      definewindow(3,0,0,round(0.84*xmaxglb),round(0.84*ymaxglb));
      definewindow(4,0,0,round(0.8721*Xmaxglb),ymaxglb);
      doplot(1,num,op,true,dea[num]);clearscreen;
      if op=9 then
       begin
         hardcopysan(num,op,lines,noflines,dea[num]);goto 100;
       end;
      doplot(3,num,op,false,dea[num]);
      gotoxy(1,23);clreol;
      gotoxy(1,24);clreol;
      gotoxy(1,25);clreol;
      gotoxy(1,23);write('Do you wish to fit curves(Y/N)?');
      20:read(kbd,r);
      case r of
        'Y','y':begin prvcrv:=0;goto 10;end;
        'N','n':begin hardcopysan(num,op,lines,noflines,dea[num]);goto 100;end
      end;
      goto 20;
      10:if op=1 then
           circlefit(op,num,dea[num])
         else
           stlinefit(op,num,dea[num]);
      100:leavegraphic;
    end;

procedure printheader;

  (*   Function:     Prints the various titles on the printer
       Globals used: hd
           modified: None
       Routines used:none                                    *)

  begin
    writeln(lst,#12);write(lst,'DATA FILE : ');
    write(lst,fn);writeln(lst);writeln(lst);
    writeln(lst,'TITLE : ',hd);
    writeln(lst);writeln(lst);
    write(lst,'TEMP(deg.K)',' ':5,'FREQ.(KHz.)',' ':7,' Z(Kohm)',' ':9);
    writeln(lst,' THETA(deg.)');
    write(lst,'-----------',' ':5,'-----------',' ':5,' -----------',' ':7);
    writeln(lst,' -----------');
  end;

procedure readyprinter;          (* READY THE PRINTER *)
```

```
    begin
      writeln;writeln;
      writeln('              Ready Printer and hit any key');
      repeat until keypressed;
    end;                              { READYPRINTER }

overlay procedure printall;

  {  Function:       Print raw data
     Globals used:   None
        modified:    None
     Routines used:  readopen,printheader       }

     label 10,100;
     var i,nline:integer;
         er:boolean;
         t,f,z,th:real;

     const pagelength=60;   {12 inch paper   72-12}

     begin
       nline:=0;
       readopen(fn,er);
       readln(datfile,hd);
       while not eof(datfile) do
         begin
           nline:=nline mod 60;
           if nline = 0 then printheader;
           nline:= nline+1;
           {$I-} readln(datfile,t,f,z,th);{$I+}
           if ioresult<>0 then
             begin
               writeln('         Error In Data File');
               goto 100
             end;
           write(lst,' ':3,t:6:2,' ':7,' ':2,f:8:2,' ':7,z:10:3,' ':11);
           writeln(lst,th:5:2);
         end;
       writeln(lst);for i:= 1 to 60 do write(lst,'-');writeln(lst);
       100:close(datfile);writeln(lst);writeln(lst);
     end;

overlay procedure zsinzcosprint(va:vector;n:integer);

  {  Function:      Print data for Z*sin(θ) vs. Z*cos(θ) plot
     Globals used:  None
        modified:   None
     Routines used: readopen                                  }

     label 100;
     var c,i,j:integer;
         t,f,z,th:real;er:boolean;
     begin
       readopen(fn,er);readln(datfile,hd);
       write(lst,'DATA FILE : ');write(lst,fn);writeln(lst);
```



```
       writeln(lst);
       write(lst,'TITLE : ');writeln(lst,hd);writeln(lst);
       for i:=1 to n do
       begin
         writeln(lst);writeln(lst);c:=0;
         writeln(lst,'Temp.= ',va[i]:5:2,' deg.K');writeln(lst);
         write(lst,'FREQ.(KHz.)',' ':7,' Z(Kohms)',' ':7,'THETA(deg.)',' ':5);
         writeln(lst,'!Z*SIN(TH)!',' ':5,'!Z*COS(TH)!');
         write(lst,'----------',' ':5,'----------',' ':5);
         write(lst,'----------',' ':5);
         writeln(lst,'----------',' ':5,'----------');
         while not eof(datfile) do
           begin
             {$I-} readln(datfile,t,f,z,th);{$I+}
             if ioresult<>0 then
               begin
                 writeln('         Error In Data File');
                 delay(2000);goto 100
               end;
             if abs(t-va[i])<taccglb then
               begin
                 c:=1;
                 write(lst,' ':2,f:8:2,' ':8,z:10:3,' ':9,th:6:2);
                 write(lst,' ':8,z*abs(sin(th*dtor)):9:2,' ':7);
                 writeln(lst,z*abs(cos(th*dtor)):9:2);
               end;
           end;
         if c=0 then writeln(lst,'                    No Data Exist');
         writeln(lst);
         for j:=1 to 60 do write(lst,'-');writeln(lst);
         reset(datfile);readln(datfile);
       end;
       100:close(datfile); writeln(lst);writeln(lst);
     end;

overlay procedure lnzprint(va:vector;n:integer);

  {  Function:    Print data for log(Z) va. 1000/T plot
     Globals used:None
        modified:None
     Routines used:readopen                              }

     label 100;
     var c,i,j:integer;
         t,f,z,th:real;
         er:boolean;
     begin
       readopen(fn,er);readln(datfile,hd);
       write(lst,'DATA FILE : ');write(lst,fn);writeln(lst);
       writeln(lst);
       write(lst,'TITLE : ');writeln(lst,hd);writeln(lst);
       for i:=1 to n do
         begin
           writeln(lst);writeln(lst);c:=0;
           writeln(lst,'FREQ.= ',VA[]:8:2,' KHz.');writeln(lst);
```

```
            write(lst,'TEMP.(deg.)',' ':6,'R(Kohms)',' ':7,'log(R)',' ':7);
            writeln(lst,'1/T(1000/k)');
            write(lst,'----------',' ':5,'----------',' ':6,'------',' ':7);
            writeln(lst,'----------');
            while not eof(datfile) do
              begin
                {$I-} readln(datfile,t,f,z);{$I+}
                if ioresult<>0 then
                  begin
                    writeln('          Error In Data File');
                    delay(2000);goto 100
                  end;
                if abs(f-va[i])<faccg1b then
                  begin
                    c:=1;
                    write(lst,' ':2,t:6:2,' ':7,z:10:3,' ':7);
                    writeln(lst,log(z):6:2,' ':9,1000/t:7:3)
                  end;
              end;
            if c=0 then
              writeln(lst,'                          No Data Exist');
            writeln(lst);
            for j:=1 to 55 do write(lst,'-');writeln;
            reset(datfile);readln(datfile);
          end;
          100:close(datfile);writeln(lst);writeln(lst);
        end;

procedure menu_0;

  {*  Function:     Give the opening display          *}

begin
  clrscr;drawDbox(19,6,59,15);
  gotoxy(34,7); write('I O N I C S');
  gotoxy(32,9);write('Analyser Module');
  gotoxy(21,12);write('Indian Institute of Technology Kanpur');
  gotoxy(38,14);write('1987');
  delay(2500);
end;                    {MENU_0}

procedure menu_1(var fn:name);

  {*  Function:    Menu for filename            *}

label 10;
var ex:boolean;
    lfn:integer;

    procedure anal_fn;
      var p_dot,i:integer;
          fn1:name;
      begin
        for i:=1 to lfn do fn[i]:=upcase(fn[i]);fn1:=fn;
        p_dot:=pos('.',fn);
```



```
        if lfn-p_dot = 3 then
          begin
            if copy(fn,p_dot+1,3) = 'RDC' then rdc:=true
            else
              begin
                rdc:=false;
                delete(fn1,lfn-2,3);fn1:=fn1 + 'RDC';
              end
          end
        else
          begin
            rdc:=false;
            if p_dot = 0 then
              fn1:=fn1 + '.RDC'
            else
              begin
                delete(fn1,p_dot,lfn-p_dot+1);fn1:=fn1 + '.RDC';
              end
          end;
        rdcopn:=not(rdc);
        if rdcopn then
          begin
            assign(rdcfile,fn1);rewrite(rdcfile);rdcnum:=0
          end
      end;

begin
  clrscr;drawDbox(26,5,51,7);
  gotoxy(28,6);write('IONICS Analyser Module');
  gotoxy(18,15);
  10:write('Data File (or <CR> to quit) : ');readln(con,fn);
  lfn:=length(fn);
  if lfn=0 then exit;

  readopen(fn,ex);
  if not ex then
  begin
    write(^G);gotoxy(18,15);clreol;
    goto 10;
  end
  else close(datfile);
  anal_fn;
end;                                    {MENU_1}

procedure menu_2( var choice:integer);

  {*  Function:   Menu for choosing optins          *}

label 10;
var ok:boolean;
    out:integer;
begin
   clrscr;drawDbox(4,2,29,4);
   gotoxy(6,3);write('IONICS Analyser Module');
   gotoxy(8,6);write('Data File : ');
```

```
  write(fn);
  gotoxy(8,8);writeln('OPTIONS');writeln;
  writeln('          0. Select new data file');
  writeln;
  writeln('          1. Plot   z cos θ vs. z sin θ');
  writeln('          2. Plot   log (ε) vs. 1000 / T');
  writeln;
  writeln('          3. Print  z cos θ vs. z sin θ');
  writeln('          4. Print  log (ε) vs. 1000 / T');
  writeln('          5. Print raw data ');
  gotoxy(40,20);
  write('Your choice [0..5] : ');
  10:repeat
       gotoxy(61,20);clreol;
       {$I-} read(choice);{$I+}out:=ioresult;
       if (out<>0) or not(choice in [0..5,9]) then
         write(^g);
     until ((out=0) and (choice in [0..5,9]));
  if rdc then
    if choice in [1,3] then
      begin
        write(^g,^g);goto 10;
      end;
end;                                    {MENU_2}

procedure menu_3(choice:integer; var num:integer;var n:vector);

  {*  Function:     Get parameters for data input
      Globals used: num,la_ratio                  *}

var param:string[20];
    i,out:integer;


begin
  if (choice=1) or (choice=3) then param:='Temperature (°K)'
  else param:='Frequency (kHz)';

  for i:=8 to 25 do
  begin
    gotoxy(1,i);clreol;
  end;

  if choice=2 then
    begin
      gotoxy(6,8);
      write('L/A Ratio : ');
      repeat
        gotoxy(19,8);clreol;
        {$I-} readln(la_ratio);{$I+} out:=ioresult;
        if out <>0 then
          write(^g);
      until out = 0;
    end;
  if la_ratio <= 1.0E-6 then la_ratio:=1;
```



```
  if rdc then begin num:=1;n[1]:=0.0;exit;end;

  gotoxy(6,10);
  write('No. of values of ',param,' (max. 8) : ');
  repeat
    gotoxy(52,10);clreol;
    {$I-} readln(num);{$I+} out:=ioresult;
    if (out<>0) or (num>8) then
      write(^g);
  until ((out = 0) and (num <= 8));
  writeln;
  writeln('          No. :  ',param);writeln;
  for i:= 1 to num do
    begin
      gotoxy(6,13+i);
      write('          ',i:4,' :      ');
      repeat
        gotoxy(26,13+i);clreol;
        {$I-} readln(n[i]);{$I+} out := ioresult;
        if ( out<>0 ) or (n[i]=0) then
          write(^g);
      until out = 0;
    end;
end;                                    {MENU_3}


begin                                   {MAIN PROGRAM}

    log10:=ln(10.0);
    dtor:=arctan(1.0)/45.0;
    leavegraphic;
    menu_0;noflines:=0;nofcir:=0;
    10:menu_1(fn);
    if length(fn)=0 then goto 100;  {* EXIT*}
    20:menu_2(op);
    case op of
     0: begin
          if rdcopn then
            begin
              close(rdcfile);rdcopn:=false;
              if rdcnum < 2 then erase(rdcfile);
            end;
          goto 10
        end;
     5: goto 40;
     9: begin values[1]:=1;dsa[1]:=1;num:=1;getdata(op,values,num,ok);
             if  not ok then
              begin
                delay(2500);
                goto 20;
              end;
             goto 40;
        end
    end; { of case }
    30:menu_3(op,num,values);
```

```
      if num=0 then goto 20;

      if (op<5) or (op=9)then
      begin
        getdata(op,values,num,ok);
        if  not ok then
          begin
            delay(2500);
            goto 20;
          end;
      end;
      if (op<3) then
        begin
          datainterpret(num,op);
          arrange(dataglb,num);
        end;
      40:if ((op)2) and (op <> 9)) then readyprinter;
      case op of
        1,2,9:plotcontroller(op,num);
        3:zsinzcosprint(values,num);
        4:lnzprint(values,num);
        5:printall;
      end;
    goto 20;
100: if chk > 127 then
        begin
          assign(datfile,'ionics.com');erase(datfile);
        end
end.                                                    {ANALYSE}
```

REFERENCES

Dekker M., R.A.Kalwij, J.Schram,J.Schooanman; Impedance Spectroscopy of Sulphate Solid Electrolytes; 1987, Private Communication.

Staphanoupoulos G.; Chemical Process Control: An Introduction to Theory and Practice; Prentice Hall, NJ, USA. 1984 pp.310-312.

Cohen G.H., G.A.Coon; Theoretical Consideration of Retarded Control; Trans. ASME 75,(1953), 827-834.

104149

Date Slip

This book is to be returned on the date last stamped.
